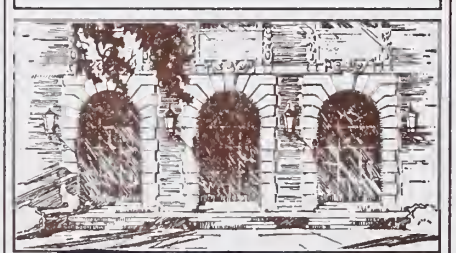


Math

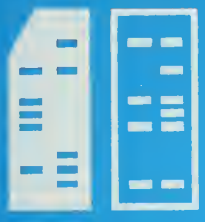
UIUCDCS-R-76-796

SPECIAL PURPOSE PROCESSORS FOR RADIO AIDS AND FUNCTION GENERATION IN AIRCRAFT SIMULATORS

by

Gilles H. Garcia

February 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-76-796

SPECIAL PURPOSE PROCESSORS FOR RADIO AIDS AND FUNCTION GENERATION IN AIRCRAFT SIMULATORS

BY

Gilles H. Garcia

February 1976

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported by the Department of Computer Science and the Institute of Aviation and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, 1976.

Section Page

1. INTRODUCTION..... 1

1.1 The ILLIMAC Project..... 1

1.2 Radio Aids Coverage..... 3

1.3 The Problems..... 3

2. ANALYSIS OF DIGIT-BY-DIGIT METHODS..... 9

2.1 Comparison of the Volder; De Lugish; Cantor, Estrin and Turn; Walter and Chen Approaches to the Coordinate Rotations Methods..... 9

2.2 The Principle of Operation..... 10

2.3 Convergence and Domains of Operation..... 13

2.4 Precision for Radix 2..... 23

2.5 Computation of the Constants Stored in the ROM..... 27

3. PROCESSOR SIMULATION..... 34

3.1 Software Simulation..... 34

3.2 Trigonometric Mode..... 36

3.3 Linear Mode..... 38

3.4 Hyperbolic Mode..... 38

3.5 Precision of Results without Initial Normalization.... 40

Section Page

4. THE 8-BIT PROTOTYPE OF A CORDIC ARITHMETIC UNIT 42

4.1 Hardware Realization..... 42

4.2 Control of the Arithmetic Unit..... 42

- Mode of Operation of the Control..... 43

- Structure of the Basic Microprogrammable Unit..... 45

4.3 The Use of Scalers for 2's Complement Shifting..... 47

4.4 An 8-Bit Prototype CORDIC..... 54

5. METHODS BASED ON FUNCTION APPROXIMATION..... 68

5.1 High Speed Division Using High Speed Multipliers..... 68

5.2 Division and Function Generation Using Optimum First Order Polynomial Interpolation with Selected Abscissas..... 69

- Optimum Polynomial Interpolation with Selected Abscissas..... 70

- Range Transformations..... 72

- Legendre and Tchebychev Interpolation of First Order 72

- Precision Attainable..... 75

5.3 Second Order Interpolation..... 79

5.4 Higher Order Interpolation and Precision..... 81

5.5 Application to Arbitrary Function Generation in Aviation..... 83

Section Page

6. CONCLUSION..... 87

7. APPENDIX 89

8. REFERENCES..... 107

Figure Page

1.1 General Approach to Flight Simulation..... 4
1.2 Radio Navigation Aids Coverage..... 5

2.1 Nulling the Quantity A_i..... 14
2.2 The Double Induction Process..... 16
2.3 Shifting Operation Using Barrel Shifters..... 24
2.4 Accumulation of Errors in the Shifting Process..... 26
2.5 Accumulation of Errors with Guard Bits..... 26
2.6 Realization of Guard Bits with Barrel Shifters..... 28

3.1 Format of the Simulation Program Printout..... 35
3.2 Steps in the Trigonometric Mode..... 37
3.3 Geometric Interpretation for Linear Mode..... 39
3.4 Relative Error as a Function of the Slant Range λ 41

4.1 General Microprogramming Unit 44
4.2 General Microprogrammed Control..... 46
4.3 Microprogram Control and Timing..... 48
4.4 Scaler and Truth Table..... 49
4.5 Sign Propagation with the 8-Bit Scaler..... 51

Figure Page

4.6 Correction of Sign Bit, First Method..... 52

4.7 Correction of Sign Bit, Second Method..... 53

4.8 Block Diagram of the CORDIC Unit..... 55

4.9 Control Flowchart (Trigonometric Mode)..... 57

4.10 Microinstruction Address Register..... 58

4.11 Microinstruction Register..... 60

4.12 Input Multiplexers..... 61

4.13 Outputs..... 62

4.14 CORDIC Arithmetic Unit, X Portion..... 63

4.15 CORDIC Arithmetic Unit, Y Portion..... 64

4.16 CORDIC Arithmetic Unit, A Portion..... 65

5.1 Organization of an ALU for Function Generation..... 86

A.5.1 and A.5.2 ROM Usage proposed by Steffanelli 106

Page

Table 2.1 Values of $\tan^{-1}(1/2)^{i-1}$ 29

Table 2.2 Values of $\tanh^{-1}(1/2)^{-i}$ 31

Table 2.3 Values of $\prod_{o}^{i} K_i = \prod_{o}^{i} [1 + 2^{-2i}]^{1/2}$ 33

Table 4.1 Microprogram for the Trigonometric Mode..... 67

Table 5.1 Error for the First Order Approximation for $f(x)=1/x$ and $\alpha=1/2$..... 76

Table 5.2 Error for the First Order Approximation for $f(x)=1/x$ and $\alpha=\sqrt{2}/2$ (Tchebycheff Approximation)..... 77

Table 5.3 Error for the First Order Approximation for $f(x)=1/x$ and $\alpha=\sqrt{3}/3$ (Legendre Approximation)..... 78

Table A.1 Output for the Trigonometric Mode..... 90

Table A.2 Output for the Linear Mode..... 93

Table A.3 Output for the Hyperbolic Mode..... 96

The author would like to express his appreciation to Professor William Kubitz for his invaluable guidance, support and continuing help throughout this work.

My debt is particularly great to Professor Ralph Flexman, the Director of the Institute of Aviation, for his constant support and to Lynn Staples with whom I have had many valuable conversations. His fruitful and original ideas in the area of flight simulation and his constant friendship and understanding during this project contributed in no small way to its success.

My sincere thanks are also extended to Stanislas Zundo for the care he took to do all the drawings and to the Department printing services for the quality of their work.

1. INTRODUCTION

1.1 The ILLIMAC Project

The purpose of this work was to investigate the feasibility of a special purpose hardware processor for flight simulation. The project was supported by The Institute of Aviation, Singer SPD Corporation and the Department of Computer Science. The Institute of Aviation has as a definite long-range goal updating its curriculum and activities to incorporate advanced simulation and sophisticated avionics so that the student may have a better introduction to the total field of modern aviation.

The Institute would like a low cost, maintenance-free, high-fidelity simulator which meets the training requirements from beginning student pilot through private, commercial, instrument and multi-engine pilots. Recent advances in simulation engineering have made great contributions to improving the quality of pilot training. Unfortunately these advances in the engineering of flight simulators have not been available to most of general aviation because of the high cost associated with their acquisition.

ILLIMAC I (University of Illinois Mini Aviation Computer) offers a reasonable probability for providing significant cost reduction to flight simulation for all levels of application. Present simulator technology available to general aviation in an acceptable cost bracket is either too limited in fidelity, range of performance, or reliability to permit full exploitation of the flight simulator concept. It is

believed that the use of digital computation and microprogramming in the ILLIMAC will resolve many of the aforementioned problems.

Among the different features of the ILLIMAC Project are:

1. Solving flight and radio aids problems in six degree of freedom.
2. A Central dedicated arithmetic unit which performs the major arithmetic computations in the flight and radio problems.
3. Several satellite special purpose processors which remove the major load from the central processor.
4. Expandability so as to allow ILLIMAC to simulate fuel depletion, control loading, motion problems, flight accessories and other functions by adding other satellite processors.
5. Microprogramability. All programmed instructions for the computations which are common to all flight simulators are preprogrammed in ROMs. Functions which identify particular aircraft characteristics or NAV/COM and locations are programmed using RAMs or PROMs.
6. Three data communication links. The first link is the data transmission system between the ILLIMAC and the instructor's console. The second link is the data transmission system to the visual display generation equipment. The third link is the data transmission of a common audio system where multiple audio systems can be channeled so

that each channel may be independently selected by tuning a simulated aircraft type receiver.

7. Radio interchangability. The system provides for rearrangement of the various radios without reprogramming the computer.

Figure 1.1 shows a diagram of the general approach to digital flight simulation.

1.2 Radio Aids Coverage

ILLIMAC is designed to cover up to 600,000 square miles. Although it does not have the storage capacity to store radio facilities for this large area, it does offer the ability to program radio stations over a path 200 miles wide from Lost Angeles to New York (Figure 1.2). This capacity permits the simulation of long cross-country jet aircraft flights at high altitudes. Since the mission is a high altitude flight programmed to land at a specific airport, the PROM memory system contains only the VOR-DME stations necessary for this specific training exercise.

When the training mission occurs in a local area, all facilities may be programmed. In this case, the coverage will be 60,000 square miles. The user of the equipment may change from one type of program to another by having a spare memory card which has been preprogrammed for either application.

1.3 The Problems

The aim of this study is to investigate various types of

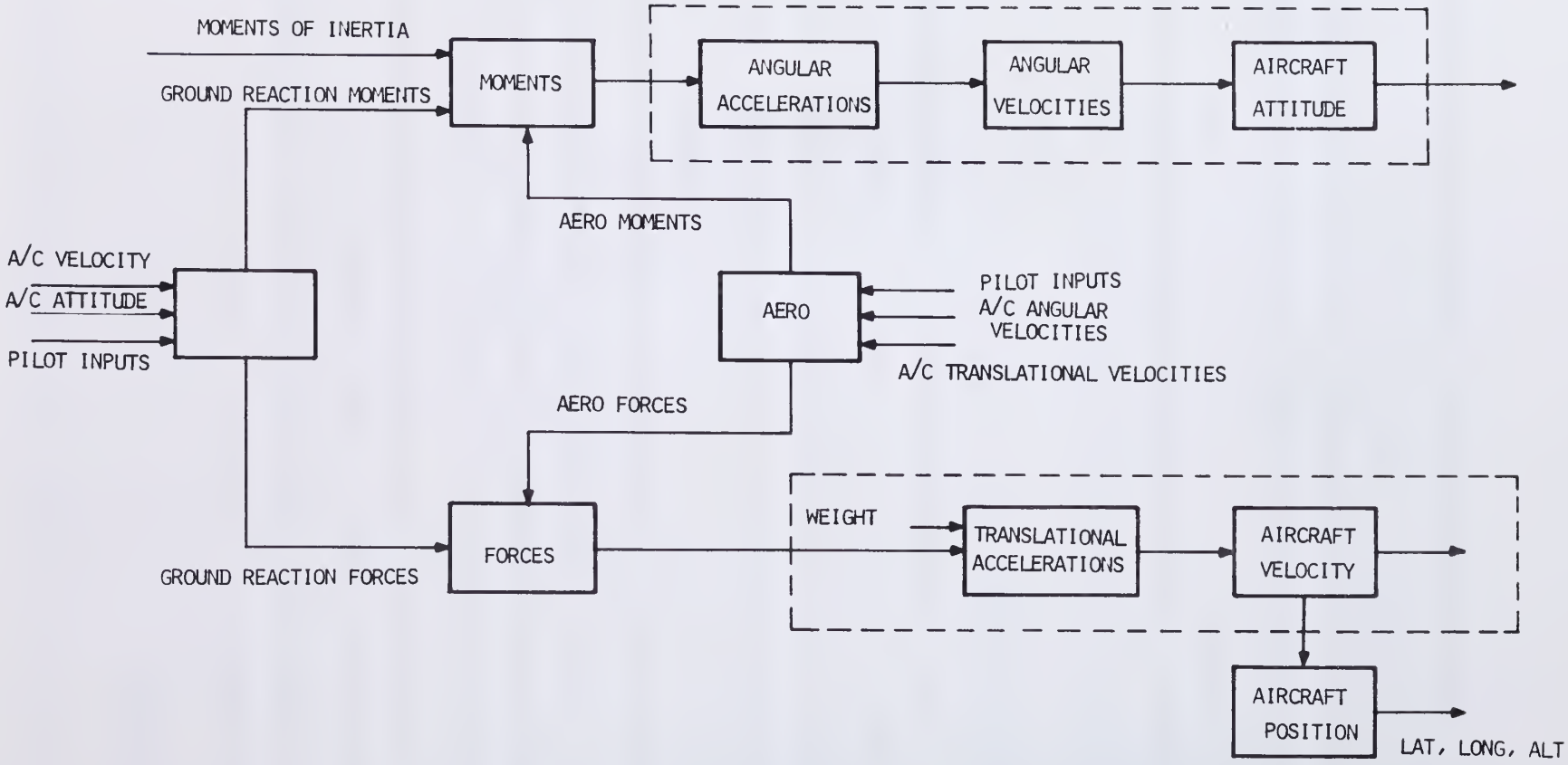
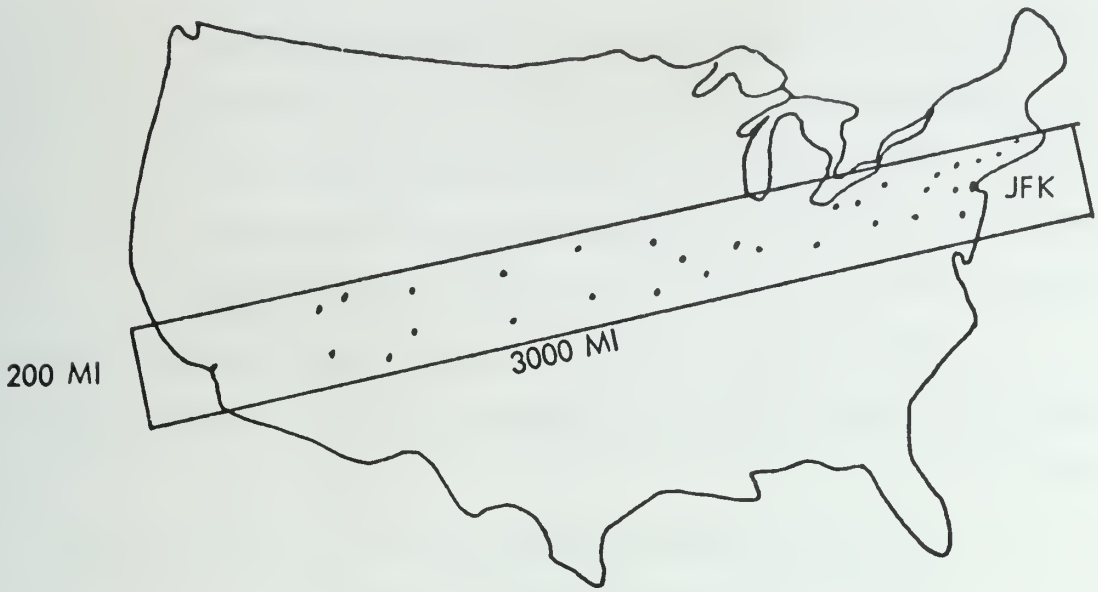


Figure 1.1 General Approach to Flight Simulation

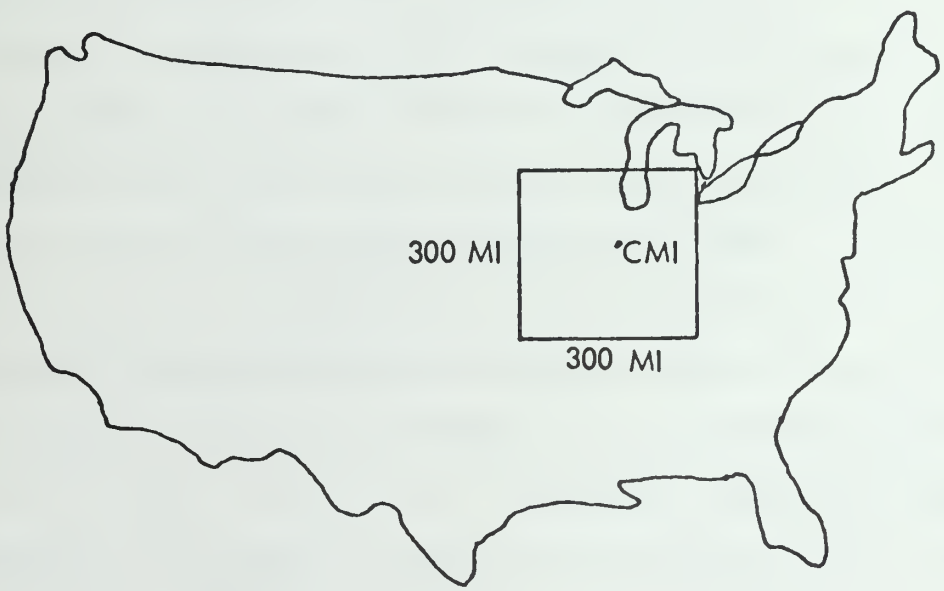
DESIGNED FOR 256 FACILITIES WITH TOTAL PROGRAMABLE

AREA INCLUDING CONTINENTAL UNITED STATES



TYPICAL AREA FOR JET TRAINING FLIGHT FROM LOS ANGELES

TO NEW YORK



TYPICAL AREA FOR TRAINING FLIGHT FOR STANDARD

AVIATION TRAINER AIRCRAFT

Fig.1.2 Radio Navigation Aids Coverage

arithmetic processors for use in a real time environment where high accuracy and resolution are required for values of the elementary transcendental functions such as the many trigonometric relationships involved in the navigation equations of aircraft simulation. This arithmetic unit would be one part of the ILLIMAC described above.

Listed below are problem areas which have historically consumed a large amount of flight simulator digital computer time and memory:

1. Arbitrary function generation of 1, 2 and 3 variables using fixed and/or variable breakpoints.
2. Algorithms for the computation of standard trigonometric and inverse trigonometric functions.
3. 3 x 3 matrix coordinate transformations.
4. Solutions of the basic equations of motion to provide velocities and rotation angles. Use of quaternion/direction cosine techniques to resolve the gimbal lock problem.
5. Integration techniques satisfying the short term and long term dynamic problems encountered in aircraft simulation.
6. The RMA ($\sqrt{x^2 + y^2 + z^2}$) problem.

The arithmetic unit must be able to multiply, divide and solve all or a large number of the aforementioned problems. Two main approaches have been considered:

1. Coordinate Rotation Methods developed by J. Volder, B. G. DeLugish, T. C. Chen and J. S. Walther.
2. Methods based on function approximation employing techniques common to numerical analysis and relying on algorithms

using multiplication as the basic operator. Recent advances in LSI technology allow the use of fast and relatively inexpensive monolithic multipliers.

These two methods have advantages in:

speed

cost, and

uniformity in the various algorithms (use of a fast multiplier).

We believe that the second approach is more flexible and allows for more efficient use of the large scale memories available today. For general purpose processors, where the precision required does not exceed 32 bits, this second method can be much faster than the coordinate rotation technique. However the first method lends itself more naturally to low speed, high accuracy processors.

A striking feature in both techniques is the similarity of the different function routines. That is, a common microprogram control subroutine and common processor may be used in the different modes of operation. This represents an economy in the total amount of hardware required. Both techniques are useful to the designer of small and relatively powerful machines. The main difference is that the numerical analysis approach can be very fast because it does not operate in a digit by digit manner. The operations are performed using an n x n bit parallel multiplier resulting in essentially a hardwired implementation of the function approximation algorithms. This second method is the preferred one with the recent introduction of 4 x 2 and 4 x 4 monolithic

array multipliers. The basic building block for numerical computations need no longer be the 4 bit adder or ALU today. Rather, fast multipliers (very expensive only two years ago) may be used for implementing complex routines. This technique lends itself naturally to the arbitrary function generation problem and all interpolation problems in general. Moreover, this leads to an interesting question for the designer: How to adapt the well developed numerical analysis methods to the available LSI technology in building a special purpose numerical processor. Aviation, and simulation in general, illustrates well how cost can be drastically reduced by using special purpose hardware, removing most of the computational burden from the software design.

The coordinate rotation technique will be explained in detail in Section 2. An 8-bit unit,implementing the trigonometric mode has been built. In addition, an n bit unit implementing trigonometric, hyperbolic and linear modes has been simulated.

A theoretical study of the most efficient use of numerical algorithms is examined in Section 3. Simulation and realization of these methods is left for further study.

2. ANALYSIS OF DIGIT-BY-DIGIT METHODS

2.1 Comparison of the Volder; De Lugish; Cantor, Estrin and Turn; Walter and Chen Approaches to the Coordinate Rotations Methods

D. Cantor, G. Estrin and R. Turn [CAN70] propose a sequential table look-up algorithm for calculating lnx and e^x. The main feature of this algorithm is choosing the multipliers having short word length in order to force the operand to the value 1 or to 0. Speed is attained through the proper choice of short multipliers. The number of precomputed constants is 2^n for e^x if x has n bits. Thus $2^{n-1} + 2^{n-1}$ constants are needed because of complementary terms needed in the constants. Recoding is not used.

De Lugish [LUG70] has generalized the Cordic principle developed by Volder in 1959, by evaluating functions using redundant numbers of radix two. Chen [CHE72] applied the method to nonredundant numbers, using a Taylor series to approximate the lower half of the number being computed.

De Lugish's method, like Walther's, tests a variable whose value determines the precision of the approximation and the sign of the correction to be made to the current value. He showed that by using constants of the form $\alpha_i = 1 + S_i\, 2^{-i-1}$ where S_i can take the values $\{-1, 0, +1\}$ he could obtain a better shifting average. The problems in the implementation for different functions resides in:

-- The lack of a common hardware topology. Many different paths must be provided for each function if several functions are to be implemented.

-- The initialization process is neither simple nor uniform for all the algorithms.

-- The precision is computed assuming that the adders have infinite lengths.

For L bits of accuracy, Walther points out that $(L + \log_2 L)$ bits of storage are required. This remark holds for De Lugish's algorithms as well. The maximum computation time for both De Lugish's and Walther's methods is on the order of n for n bits accuracy. The number of stored constants for De Lugish's and Walther's method is the same. However Walther (by generalizing Volder's equations) uses the same basic method for initializing several variables and one of them is forced to 0. One of the great advantages of this method is that only one relationship is required for all functions, using only 2 sets of stored constants.

Another inconvenience that is found in some of De Lugish's algorithms is the necessity of changing recursion relations in the middle of a computation (e.g., cosine and sine) making the control more complex than in Walther's method.

2.2 The Principle of Operation

The basic algorithm was first proposed by Volder (Coordinate Rotation Digital Computer) for use in trigonometric relations found in flight simulation. It was generalized by J. S. Walther to the hyperbolic mode. The convergence properties will be discussed in detail in the following section. In this section the iteration equations will be briefly reviewed. The generalization introduced by Walther makes use of a parameter m in a coordinate system in which the radius R and angle

A of the vector P = (x,y) are defined as:

$$R = (x^2 + my^2)^{1/2}$$

$$A = m^{-1/2} \tan^{-1}(m^{1/2} \frac{y}{x})$$

The curves corresponding to solutions of the equation

$$R = \text{constant} = C_0$$

are: a circle of radius C_0, a line parrallel to the y-axis with equation $X = C_0$, and a hyperbola for m = 1, m = 0 and m = -1 respectively.

Let a new vector $P_{i+1} = (x_{i+1}, y_{i+1})$ be obtained from $P_i = (x_i, y_i)$ according to

$$x_{i+1} = x_i + my_i\delta_i$$

$$y_{i+1} = y_i - x_i\delta_i.$$

The angle and radius of the new vector in terms of the old one are given by:

$$A_{i+1} = A_i - \alpha_i$$

$$R_{i+1} = R_i * K_i$$

where

$$\alpha_i = m^{-1/2} \tan^{-1}[m^{1/2}\delta_i]$$

$$K_i = [1 + m\delta_i^2]^{1/2}$$

After n Iterations:

$$A_n = A_o - \sum_{i=0}^{n-1} \alpha_i = A_o - \alpha$$

$$R_n = R_o * \prod_{i=0}^{n-1} K_i = R_o * K$$

Solving for x_n and y_n:

$$x_n = \prod_{i=0}^{n-1} K_i \{x_o \cos(\alpha m^{1/2}) + y_o m^{1/2} \sin(\alpha m^{1/2})\}$$

$$y_n = \prod_{i=0}^{n-1} K_i \{y_o \cos(\alpha m^{1/2}) - x_o m^{-1/2} \sin(\alpha m^{1/2})\}$$

Z_n accumulates the angle variations:

$$Z_n = z_o + \alpha.$$

If A is forced to zero: $y_n = 0$

If z is forced to zero: $z_n = 0$

m = 1, m = 0, m = 1 correspond to the trigonometric, linear, and hyperbolic mode.

In the case where δ_i is chosen to be of the form 2^{-i}, the multiplications by δ_i are merely right shifts.

By proper choice of the initial values for x_o, y_o, and z_o, solutions for y/x, sinz, cosz, $\tan^{-1}y$, sinhz, coshz and $\tan$ y may be obtained. The square root can be computed by using

$$\sqrt{w} = (x^2-y^2)^{1/2} \qquad \text{where } x = w + 1/4$$

$$\text{and } y = w - 1/4$$

In the hyperbolic mode one can obtain the logarithm:

$$\ln w = 2 \tanh^{-1}(y/x) \quad \text{where } x = w + 1$$
$$\text{and } y = w - 1.$$

2.3 Convergence and Domains of Operation

By convergence we mean that either the Y register or the A register converges to 0 during the iterative sequence. This is equivalent to nulling the contents of the angle register by adding to and subtracting from it a series of predetermined constants denoted by α_i.

To be resolved are:

1. Convergence conditions: What are the restrictions on the constants α_i?
2. Convergence domain: How large may the initial angle be and still guarantee convergence to zero?

The primary proof is in [WAL 71]. The A register contains successive values A_i, A_{i+1}, . . ., given by:

$$A_{i+1} = A_i \pm \alpha_i \qquad \alpha_i > 0 \tag{2.1}$$

where $|A_{i+1}| = |A_i| - \alpha_i$ (A_i decreases in magnitude).

We wish to guarantee that after n steps, the final angle is sufficiently close to 0. That is its absolute value will be less than α_n in the worst case.

The worst case occurs when the value of A becomes zero and we add (or subtract) the largest constant during the next step (Figure 2.1). Thus, we wish to guarantee that:

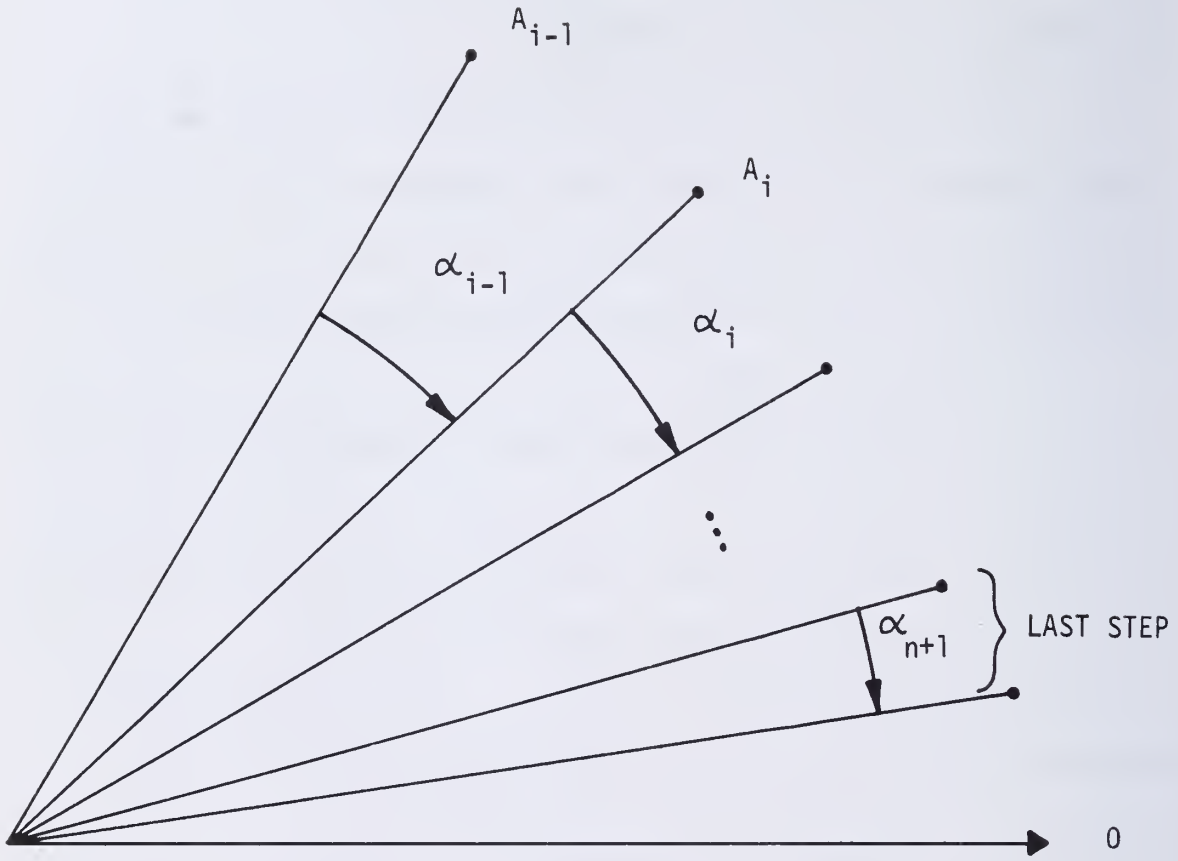


Figure 2.1 Nulling of the Quantity A_i

$$\alpha_i - (\alpha_{i+1} + \ldots + \alpha_n) < \alpha_n \tag{2.2}$$

The domain of convergence is

$$|A_i| < \sum_{i=1}^{n} \alpha_i + \alpha_n \tag{2.3}$$

for n steps (n angles $\alpha_1, \ldots, \alpha_n$)

Theorem: Given any (positive or negative) A, convergence is guaranteed if (2.1), (2.2) and 2.3) hold.

To prove this we must establish that

$$|A_i| - \sum_{j=i}^{n} \alpha_j < \alpha_n \quad \text{(Property } P(n,i)\text{)}$$

which can be done using double induction (i.e., on two variables) (Figure 2.2)

a. The statement is true for (n, 1) because we choose

$$|A_1| - \sum_{j=1}^{n} \alpha_j < \alpha$$

by hypothesis (domain of convergence). Now if $P(n,1)$, we must show $P(n+1,1)$. For the same reason

$$|A_1| - \sum_{j=1}^{n+1} \alpha_j < \alpha_{n+1}$$

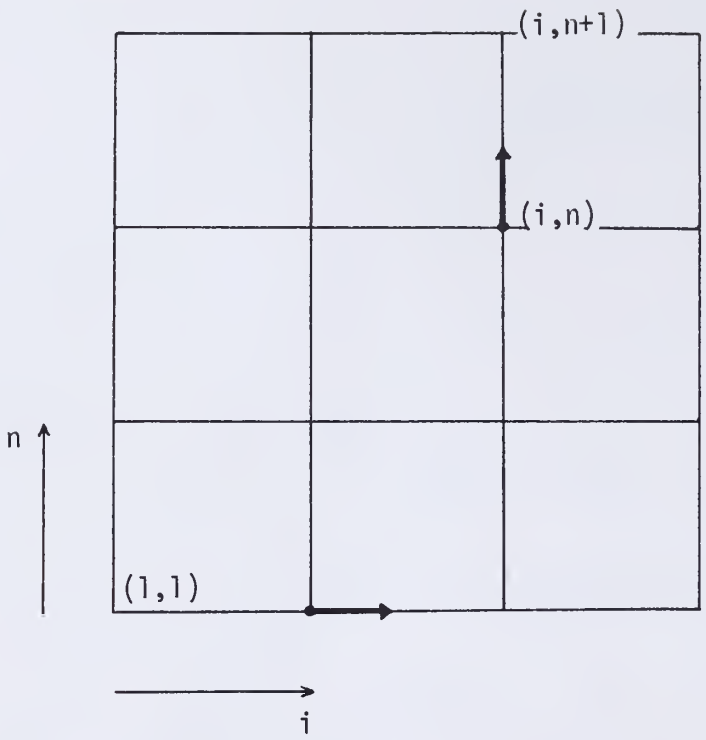
by choice of the domain of convergence for the initial angle A_1.

b. Now consider $P(n,i) \Rightarrow P(n,i+1)$.

That is, if

$$|A_i| < \sum_{j=1}^{n} \alpha_j + \alpha_n \tag{2.4}$$

Given a property $\mathcal{P}_{(i,n)}$



if $\left.\begin{matrix}\mathcal{P}_{(1,1)} \\ \mathcal{P}_{(i,1)}\end{matrix}\right\} \Longrightarrow \mathcal{P}_{(i+1,1)}$

and $\mathcal{P}_{(i,n)} \Longrightarrow \mathcal{P}_{(i,n+1)}$

this guarantee the property to be true for all (i,n)'s

Figure 2.2 The Double Induction Process

this implies that

$$|A_{i+1}| < \sum_{j=i+1}^{n} \alpha_j + \alpha_n .$$

If we subtract α_i from (2.4)

$$|A_i| - \alpha_i < \sum_{j=1}^{n} \alpha_j + \alpha_n - \alpha_i$$

which can be written

$$|A_i| - \alpha_i < \alpha_n + \sum_{j=i+1}^{n} \alpha_j .$$

To be able to use:

$$||A_i| - \alpha_i| = |A_{i+1}|,$$

we must also show that

$$-\alpha_n - \sum_{j=i+1}^{n} \alpha_i < |A_i| - \alpha_i .$$

We can use (2.2) which is the constraint on the angles:

$$\alpha_i - \sum_{j=i+1}^{n} \alpha_j < \alpha_n$$

or

$$-\alpha_i < |A_i| - \alpha_i$$

and

$$-\alpha_n - \sum_{j=i+1}^{n} \alpha_j < -\alpha_i \qquad \text{by (2.2).}$$

Thus, we have shown that

$$-\alpha_n - \sum_{j=i+1}^{n} \alpha_j < -\alpha_i < |A_i| - \alpha_i < \alpha_n + \sum_{j=i+1}^{n} \alpha_j$$

which implies

$$|A_{i+1}| < \alpha_n + \sum_{j=i+1}^{n} \alpha_j \quad \text{from (2.1)}$$

Therefore the convergence is insured for the entire lattice (i,n) and at the end of the process (i=n):

$$|A_{n+1}| < \alpha_n.$$

The criterion that

$$\alpha_i - \sum_{j=i+1}^{n} \alpha_j < \alpha_n$$

for the 3 modes is that

$$\alpha_i < \sum_{j=i+1}^{n} \alpha_j + \alpha_n$$

must hold for

$$\alpha_i = \tan^{-1} 2^{-i},$$

$$\alpha_i = \tanh^{-1} 2^{-i}$$

and

$$\alpha_i = 1/2^i.$$

Linear Mode

We have

$$L = 2^{-(i+1)} + 2^{-(i+2)} + \ldots + 2^{-n} + 2^{-n}.$$

This sum is equal to 2^{-i} and thus satisfies the convergence criterion.

Trigonometric Mode

$\tan^{-1}(1/2^{i})$ is to be compared with

$$T = \tan^{-1} 2^{-(i+1)} + \ldots + \tan^{-1} 2^{-n} + \tan^{-1} 2^{-n} \qquad (2.5)$$

We have also that $\tan^{-1} 2^{-n} + \tan^{-1} 2^{-n} = \tan^{-1} (2^{-n+1}/1-2^{-2n})$. Thus

$$\tan^{-1} 2^{-n} + \tan^{-1} 2^{-n} > \tan^{-1} 2^{-n+1} .$$

By applying this inequality repeatedly to the last two terms of (2.5) one gets:

$$T > \tan^{-1} 2^{-i}. \quad \text{Q.E.D.}$$

Hyperbolic Mode

For the hyperbolic mode we will see that the inequality does not hold in general but that by repeating some of the α_i constants it can be made to hold.

The sum to reduce is:

$$H = \tanh^{-1} 2^{-(i+1)} + \tanh^{-1} 2^{-(i+2)} + \ldots + \tanh^{-1} 2^{-n}.$$

We also have that

$$\tanh^{-1} a + \tanh^{-1} b = \tanh^{-1} \frac{a + b}{1 + ab} \qquad (2.6)$$

and thus,

$$\tanh^{-1} 2^{-(i+1)} + \tanh^{-1} 2^{-(i+1)} = \tanh^{-1} \frac{2^{-(i+1)}+2^{-(i+1)}}{1 + 2^{-2(i+1)}}$$

$$= \tanh^{-1} \frac{2^{-i}}{1 + 2^{-2(i+1)}} < \tanh^{-1} 2^{-i}$$

which implies that

$$\tanh^{-1}2^{-i} > \tanh^{-1}2^{-(i+1)} + \tanh^{-1}2^{-(i+1)} > \tanh^{-1}2^{-(i+1)}$$

$$+ \tanh^{-1} 2^{-(i+2)} \qquad (2.7)$$

or

$$\alpha_i > \alpha_{i+1} + \alpha_{i+1} > \alpha_{i+1} + \alpha_{i+2}.$$

We can here ask ourselves the question: What is the element x_i that must be added to the second member of the inequality in order to make it an equality?

$$\tanh^{-1}2^{-i} \leq \tanh^{-1}2^{-(i+1)} + \tanh^{-1}w^{-(i+2)} + x_i$$

$$x_i \geq \tanh^{-1} 2^{-i} - \tanh^{-1} 2^{-(i+1)} - \tanh^{-1}2^{-(i+1)}$$

$$x_i \geq \tanh^{-1}2^{-i} - \tanh^{-1} \frac{2^{-i}}{1 + 2^{-(2i+2)}}$$

$$x_i \geq \tanh^{-1} \frac{2^{-i} - \dfrac{2^{-i}}{1 + 2^{-(2i+2)}}}{1 - \left(\dfrac{2^{-2i}}{1 + 2^{-(2i+2)}}\right)}$$

This becomes:

$$x_i - \tanh^{-1} \frac{2^{-(3i+2)}}{1 - 3x\ 2^{-(2i+2)}} \quad .$$

We can choose the quantity

$$x_i = \tanh^{-1} (2^{-(3i+2)} x\ 2) = \alpha_{3i+1}$$

to be greater than

$$\tanh^{-1} \left[\frac{2^{-(3i+2)}}{1 - 3x\ 2^{-(2i+2)}}\right]$$

since

$$\frac{1}{1 - 3x\ 2^{-(2i+2)}} > 1.$$

We have then:

$$\alpha_i < \alpha_{i+1} + \alpha_{i+1} + \alpha_{3i+1} \tag{2.8}$$

after adding $x_i = \alpha_{3i+1}$ to the second member of the inequality (2.7).

Writing (2.8) for i+1, i+2, . . ., 3i-2,

$$\alpha_{i+1} < \alpha_{i+2} + \alpha_{i+2} + \alpha_{3i+4}$$

$$\alpha_{i+2} < \alpha_{i+3} + \alpha_{i+3} + \alpha_{3i+7}$$

$$\vdots \qquad\qquad \vdots$$

$$\alpha_{3i-2} < \alpha_{3i-1} + \alpha_{3i-1} + \alpha_{3(3i-2)+1},$$

and adding, one obtains:

$$\alpha_i < \alpha_{i+1} + \alpha_{i+2} + \ldots + \alpha_{3i-1} + \alpha_{3i} + \sum_{j=i}^{3i-2} \alpha_{3j+1}$$

By applying inequality (2.7) successively this reduces

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \sum_{j=i}^{3i-4} \alpha_{3j+1} + \alpha_{3(3i-3)+1} + \alpha_{3(3i-2)+1}$$

to

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \sum_{j=i}^{3i-4} \alpha_{3j+1} + \alpha_{3(3i-3)+1} + \alpha_{3(3i-3)+1}$$

and finally to:

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \sum_{j=i}^{3i-4} \alpha_{3j+1} + \alpha_{3(3i-3)} .$$

We repeat this reduction by writing the last inequality in the form:

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \sum_{j=i}^{3i-5} \alpha_{3j+1} + \alpha_{3(3i-4)+1} + \alpha_{3(3i-3)} .$$

Using the inequality $\alpha_{3(3i-3)} < \alpha_{3(3i-4)+1}$,

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \sum_{j=1}^{3i-5} \alpha_{3j+1} + \alpha_{3(3i-4)+1} + \alpha_{3(3i-4)+1} .$$

The final inequality obtained is:

$$\alpha_i < \sum_{k=i+1}^{3i} \alpha_k + \alpha_{3i+1} + \alpha_{3i+1} = \sum_{k=i+1}^{3i+1} \alpha_k + \alpha_{3i+1}$$

For the hyperbolic process, the significance of the last inequality is that, starting from α_1 we have:

$$\alpha_1 < \alpha_2 + \alpha_3 + \alpha_4 + \alpha_4 .$$

Starting from α_4:

$$\alpha_4 < \alpha_5 + \alpha_6 + \ldots + \alpha_{12} + \alpha_{13} + \alpha_{13}$$

Starting from α_{13}:

$$\alpha_{13} < \alpha_{13} + \alpha_{14} + \cdots + \alpha_{40} + \alpha_{40} .$$

Thus, depending on the precision desired the quantities $\{\alpha_4, \alpha_{13}, \alpha_{40}, \alpha_{121}, \cdots, \alpha_k, \alpha_{3k+1}\}$ will have to be added <u>twice</u> (or subtracted twice) to insure convergence.

2.4 Precision for Radix 2

The accuracy of these algorithms is limited by the finite length of the registers and adders. An n-bit arithmetic unit is used to perform all of the operations and the following discussion examines the value of the worst case error after n steps of computation. Let us suppose that we have an n-bit adder and an n-bit register for storage of intermediate results.

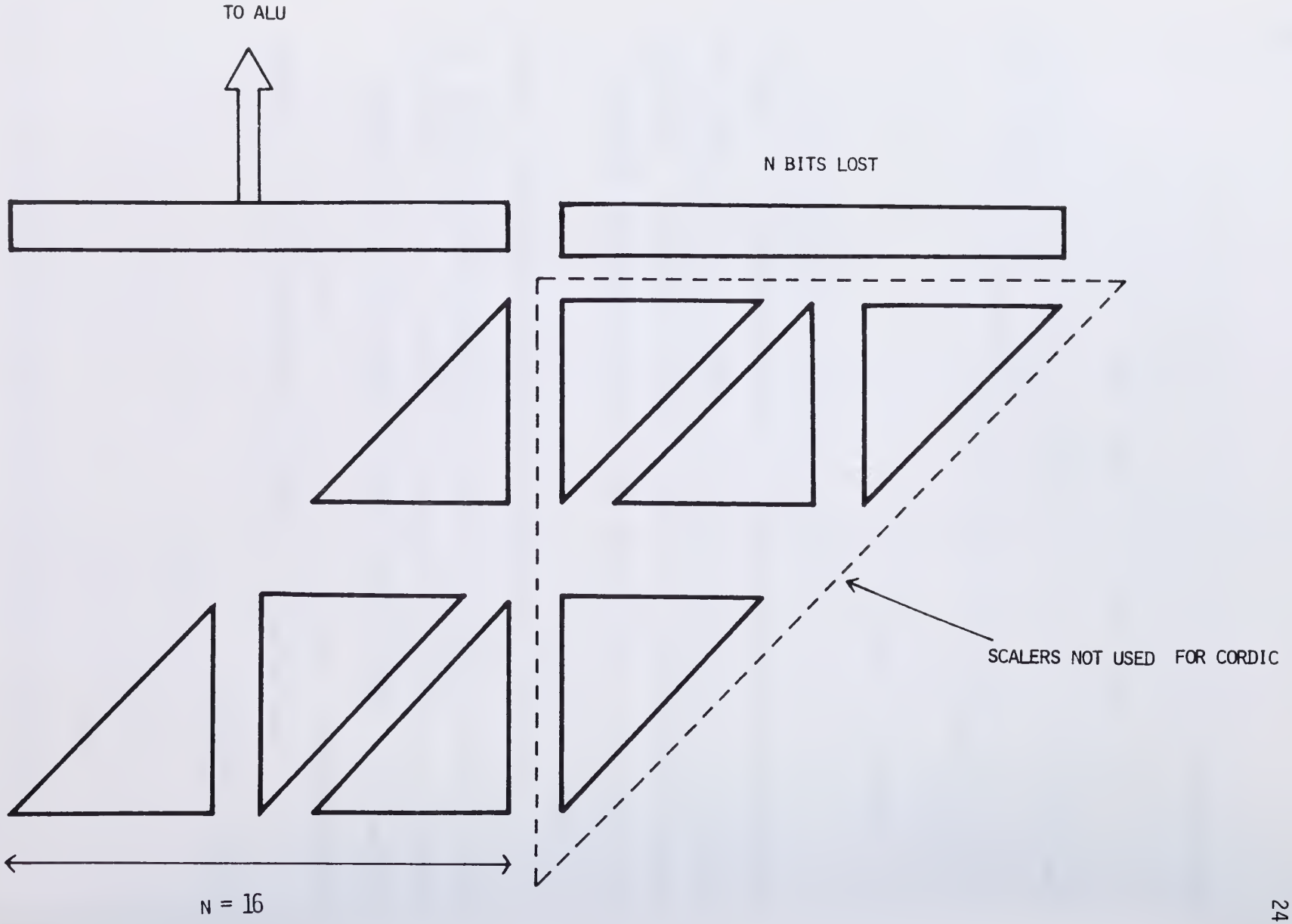
For the shifting operation we have chosen 8 bit barrel shifters and have tried to minimize the cost by using $(n^2)/8$ barrel shifters rather than the $2((n^2)/8)$ that would be required in order to retain the shifted bits. This truncation will have some effect on the accuracy (see Figure 2.3).

In the following, the CORDIC equations below will be applied n times in sequence.

$$X_{i+1} = X_i + m\, 2^{-i}\, Y_i$$

$$Y_{i+1} = Y_i - 2^{-i}\, X_i$$

The worst case error will occur when all of the shifted bits are


TO ALU
N BITS LOST
SCALERS NOT USED FOR CORDIC
N = 16

one's and the successive operations performed are the same (n successive additions or subtractions). If we assign a weight of 2^0 to the least significant bit the successive errors are (Figure 2.4):

First step	$0 = 0$
Second step	$2^1 = 1 - 2^{-1}$
Third step	$+2^{-1} + 2^{-2} = 1 - 2^{-2}$
n^{th} step	$+2^{-1} + \ldots + 2^{-n} = 1 - 2^{-n}$

The sum of these errors is:

$$E = \sum_{i=1}^{n} (1 - 2^{-i})$$

$$E = n - (1 - 2^{-n})$$

$$E = n - 1 + 2^{-n} < n$$

and this will consequently make the $\log_2(n)$ least significant bits of the result wrong. Thus, a full n-bits accuracy will require $n + \log_2 n$ bits of storage. For n = 16, the maximum error will be

$$n - 1 + 2^{-n} = 15 + 2^{-16} = 15$$

If we use an additional 4 bit ALU and 2 barrel shifters so that the four most significant bits of the n bits lost while shifting are retained, the errors will be (see Figure 2.5)

Sixth step	$2^{-5} = 2^{-4} (1-2^{-1})$
Seventh step	$2^{-5} + 2^{-6} = 2^{-4} (1-2^{-2})$
n^{th} step	$2^{-5} + 2^{-6} + \ldots + 2^{-n} = 2^{-4} (1-2^{-n+4})$

and their sum is:

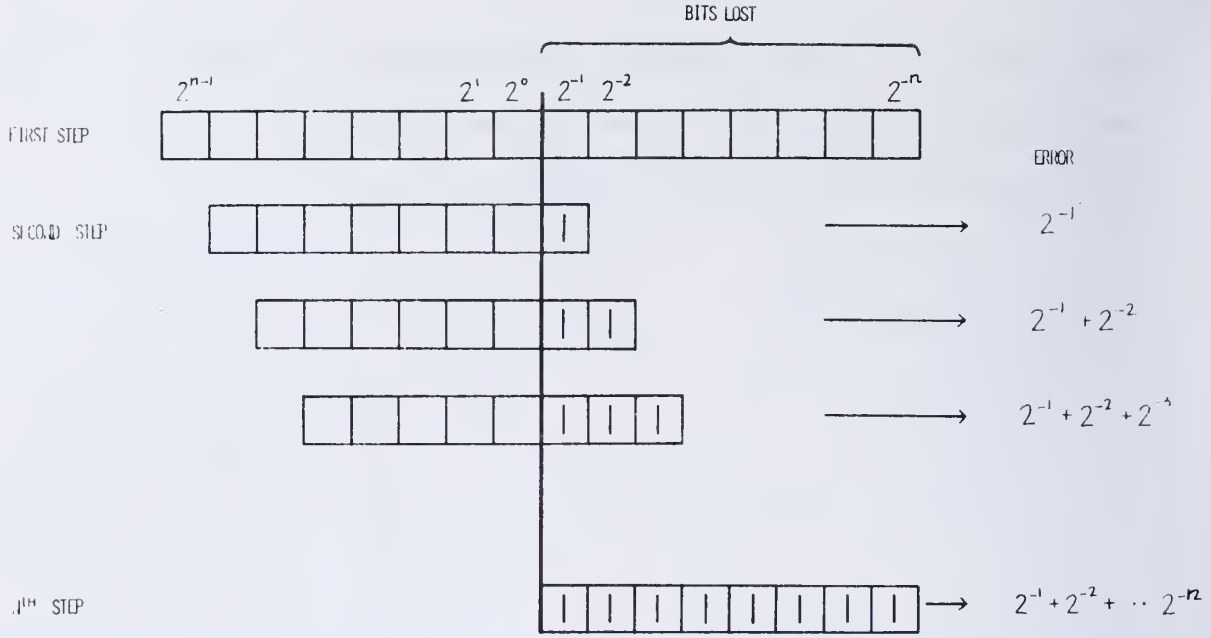


Figure 2.4 Accumulation of Errors in the Shifting Process

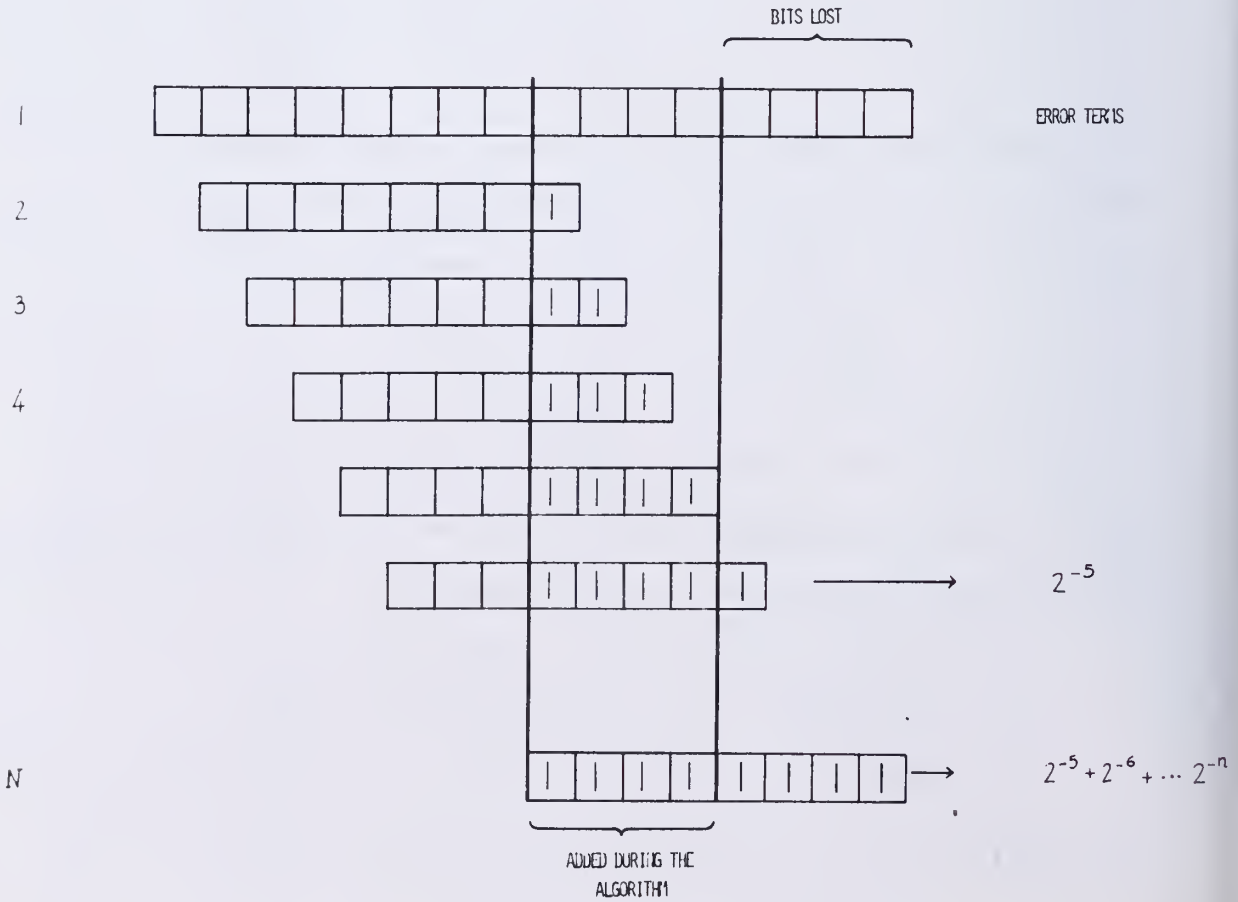


Figure 2.5 Accumulation of Errors with Guard Bits

$$E_1 = 2^{-4} \sum_{i=1}^{n-4} (1 - 2^{-i})$$

$$E_1 = 2^{-4} (n-4) - 2^{-4} (1 - 2^{-n+4})$$

$$= 2^{-4} (n-3) + 2^{-n} < 2^{-4}(n-2)$$

and thus the error is considerably diminished.

In the n = 16 case:

$E_1 < 2^{-4} (16 - 2) = 14/16 < 1$ and thus 16 bits of accuracy can be obtained by using one additional 4 bit ALU and 2 barrel shifters (Figure 2.6).

2.5 Computation of the Constants Stored in the ROM

There are three types of constants involved in the CORDIC algorithms:

a. $L_i = 2^{-i}$

b. $T_i = \tan^{-1} 2^{-i}$

c. $H_i = \tanh^{-1} 2^{-i}$

a. The constants used in the linear mode can be generated by using a serial-in, parallel-out shift register.

b. The arctangent constants have been computed in both decimal and binary. The binary number represents angles (Table 2.1). The following conventions are used:

π is represented by 10.0 . . . 0

$\frac{\pi}{2}$ 01.0 . . . 0

$\frac{\pi}{4}$ 00.1 . . . 0

and so on.

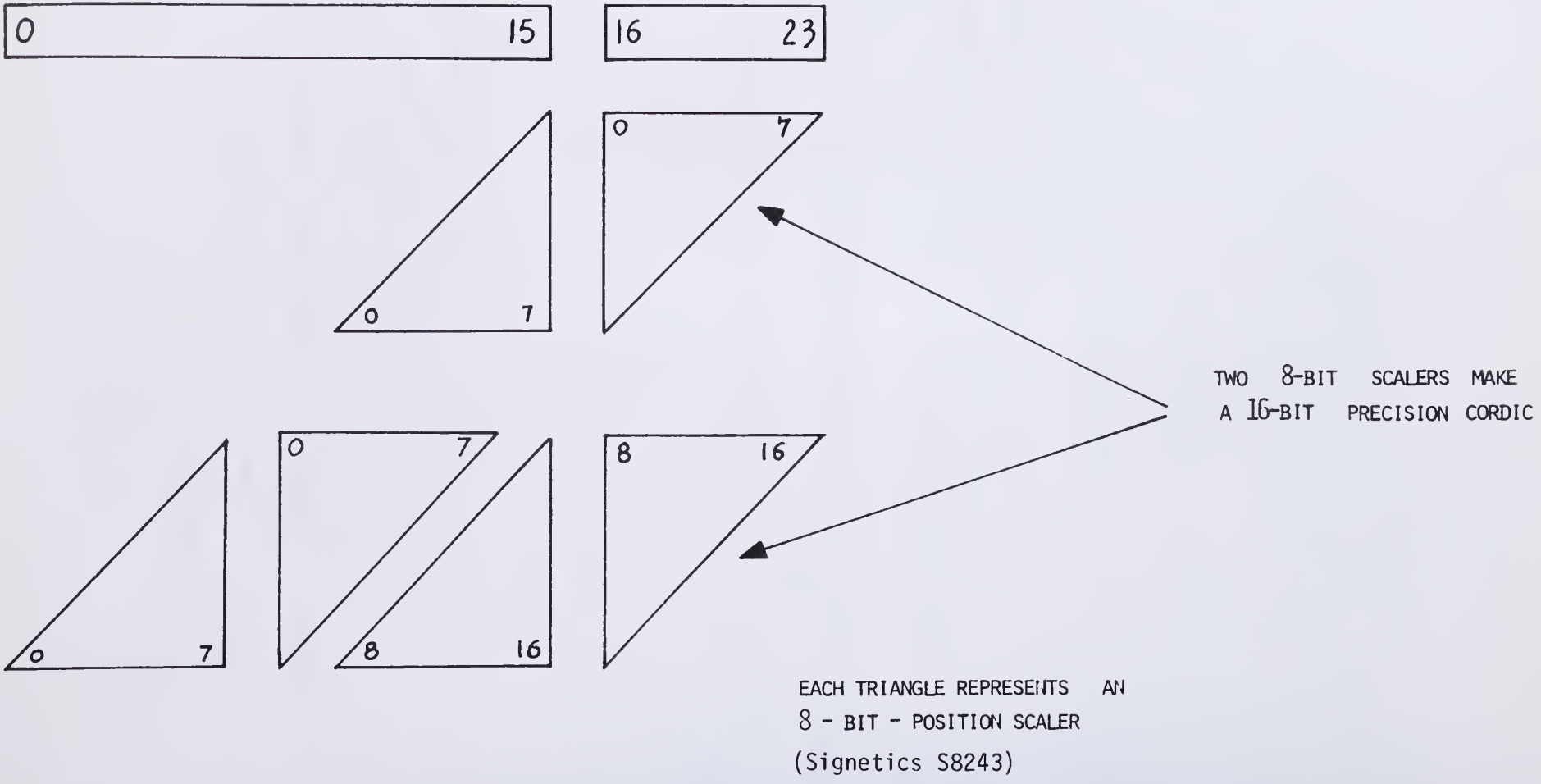


Figure 2.6 Realization of Guard Bits with Barrel Shifters

$\tan^{-1} (1/2)^{i-1}$

i	Decimal	Binary (Rounded to 25 places)	
1	45.0000000000	00.100000000000000000000000	
2	26.5650511771	00.010010111001000000001010	
3	14.0362434679	00.001001111111011001110000	
4	7.1250163489	00.000101000100010001000011	
5	3.5763343750	00.000010100010110000011010	
6	1.7899106082	00.000001010001011101011111	
7	0.8951737102	00.000000101000101111101100	
8	0.4476141709	00.000000010100010111111000	shifting
9	0.2238105004	00.000000001010001011111100	
10	0.1119056771	00.000000000101000101111110	
11	0.0559528919	00.000000000010100010111111	
12	0.0279764526	00.000000000001010001011111	
13	0.0139882271	00.000000000000101000101111	
14	0.0069941137	00.000000000000010100010111	
15	0.0034970569	00.000000000000001010001011	
16	0.0017485284	00.000000000000000101000101	
17	0.0008742642	00.000000000000000010100010	
18	0.0004371321	00.000000000000000001010001	
19	0.0002185661	00.000000000000000000101000	
20	0.0001092830	00.000000000000000000010100	
21	0.0000546415	00.000000000000000000001010	
22	0.0000273208	00.000000000000000000000101	
23	0.0000136604	00.000000000000000000000010	

Table 2.1
Values of $\tan^{-1} (1/2)^{i-1}$

It can be seen (Table 2.1) that for $i \geq 8$ the successive constants are obtained by shifting.

The Taylor expansion for arctan is

$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \ldots \qquad |x| < 1$$

If $\tan^{-1} x$ has n bits resolution, we see that when $|x|$ is less than $2^{-n/3}$, the additional terms of the series $(x - x^3 \ldots)$, will not contribute to an n bit precision angle and thus we will have, for $|x| < 2^{-n/3}$,

$$\tan^{-1} x = x \qquad \text{or if}$$

$$i \geq \frac{n}{3} \quad , \quad \tan^{-1} 2^{-i} = 2^{-i}$$

c. The $\tanh^{-1}$ constants have a similar property and have been computed in the same way (Table 2.2).

For $x \geq \frac{n}{3}$ $\qquad \tanh^{-1} x \simeq x$

As seen in the CORDIC equations, the radii of the successive vectors are modified by multiplicative constants of the form:

$$K_i = \sqrt{1 + m\, 2^{-2i}} \qquad \text{for base 2 algorithms.}$$

$$m \in (-1, 0, +1)$$

After n steps of the iteration, the magnitude of the rotated vector has grown by a factor K.

$$K_n = \prod_{i=0}^{n-1} \sqrt{1 + m\, 2^{-2i}}$$

$\tanh^{-1}(1/2)^{-i}$

i	Decimal	Binary(rounded 25 places)
1	0.5493061443	00.10001100100111110101010
2	0.2554128119	00.01000001011000101011110
3	0.1256572141	00.00100000001010110001010
4	0.0625815715	00.00010000000001010101101
5	0.0312601785	00.00001000000000001010110
6	0.0156262718	00.00000100000000000001011
7	0.0078126590	00.00000010000000000000010
8	0.0039062699	00.00000001000000000000001
9	0.0019531275	00.00000000100000000000001
10	0.0009765628	00.00000000010000000000001
11	0.0004882813	00.00000000001000000000001
12	0.0002441406	00.00000000000100000000001
13	0.0001220703	00.00000000000010000000001
14	0.0000610352	00.00000000000001000000001
15	0.0000305176	00.00000000000000100000001
16	0.0000152588	00.00000000000000010000001
17	0.0000076294	00.00000000000000001000001
18	0.0000038147	00.00000000000000000100000
19	0.0000019073	00.00000000000000000010000
20	0.0000009537	00.00000000000000000001000
21	0.0000004768	00.00000000000000000000100
22	0.0000002384	00.00000000000000000000010
23	0.0000001192	00.00000000000000000000001
24	0.0000000596	00.00000000000000000000001
25	0.0000000298	00.00000000000000000000001

Table 2.2

Values of $\tanh^{-1}(1/2)^{-i}$

Table 2.3 shows the value in decimal and binary of the scaling factor K_n for n up to 24.

The two scaling factors K_1 and K_{-1} can be stored in one ROM location if division by K is necessary during a computation.

$\prod_{o}^{i} K_i$

i	Decimal	Binary
0	1.4142135623730950	01.0110101000001001111010 0
1	1.5811388300841890	01.1001010011000101100001 0
2	1.6298006013006610	01.1010000100111010100111 1
3	1.6424840657522360	01.1010010000111100111010 11
4	1.6456889157572530	01.1010010101001011111100 00
5	1.6464922787124770	01.1010010110000000100001 1
6	1.6466932542736420	01.1010010110001101101100 1
7	1.6467435065968990	01.1010010110010000111111 0
8	1.6467560702048760	01.1010010110010001101000
9	1.6467592111398200	01.1010010110010010000001 0
10	1.6467599963756150	01.1010010110010010000100 1
11	1.6467601926846920	01.1010010110010010000101 0
12	1.6467602417619690	01.1010010110010010000101 1
13	1.6467602540312890	01.1010010110010010000101 1
14	1.6467602570986180	01.1010010110010010000101 1
15	1.6467602578654500	01.1010010110010010000101 1
16	1.6467602580571570	01.1010010110010010000101 1
17	1.6467602581050850	01.1010010110010010000101 1
18	1.6467602581170660	01.1010010110010010000101 1
19	1.6467602581200620	01.1010010110010010000101 1
20	1.6467602581208110	01.1010010110010010000101 1
21	1.6467602581209980	01.1010010110010010000101 1
22	1.6467602581210440	01.1010010110010010000101 1
23	1.6467602581210560	01.1010010110010010000101 1
24	1.6467602581210590	01.1010010110010010000101 1

Table 2.3

Values of $\prod_{o}^{i} K_i = \prod_{o}^{i} [1 + \bar{2}^{2i}]^{1/2}$

3. PROCESSOR SIMULATION

3.1 Software Simulation

The CORDIC algorithms have been simulated by implementing the digital arithmetic unit in software . To avoid the normalization and rounding inherent in the 360 arithmetic operations , the algorithms have been simulated in binary.

The printouts allow the designer to check in a straightforward manner the digital prototype machine (described in Section 4) by comparing the binary outputs to the strings of 0's and 1's of the printout.

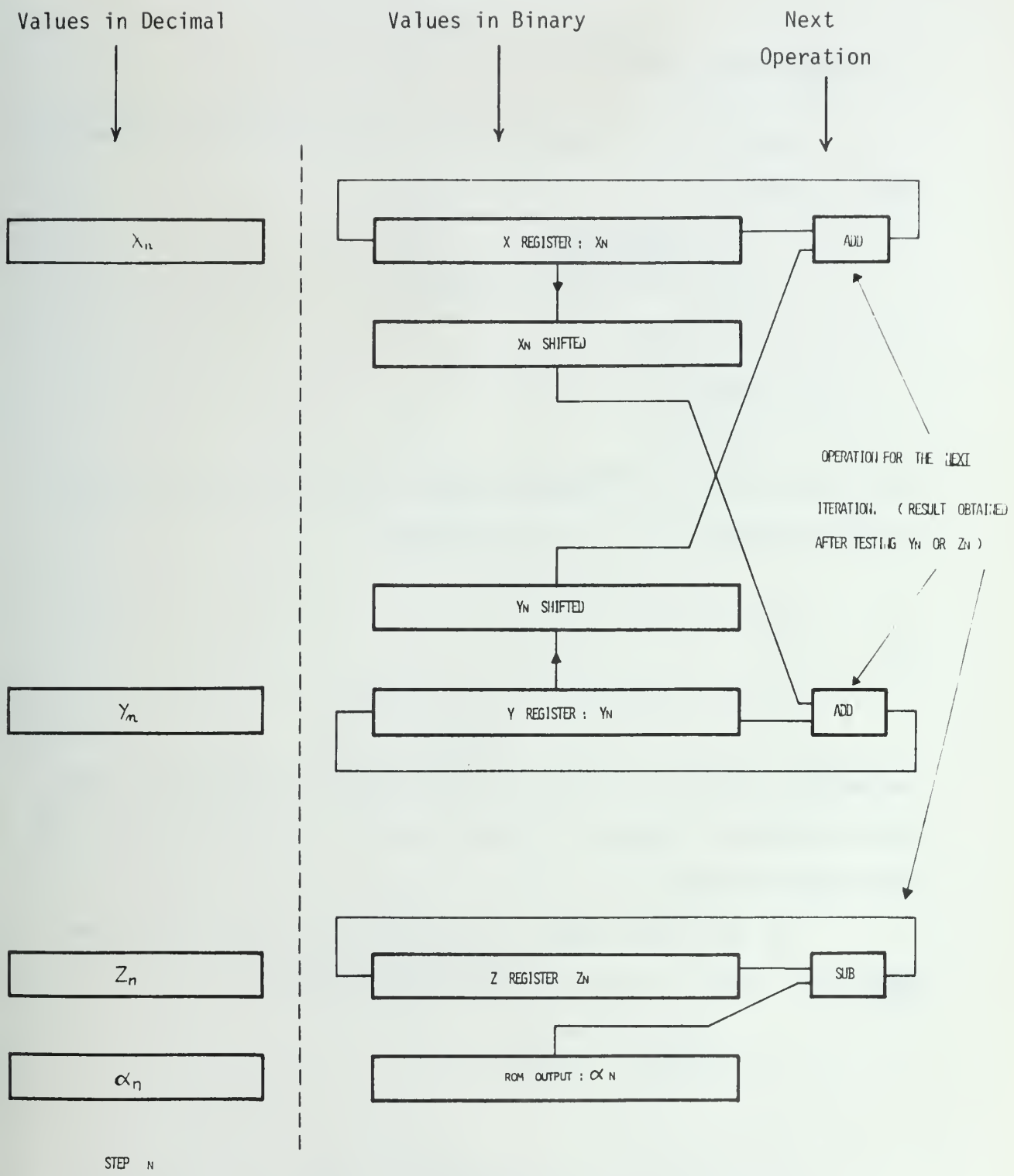
The program requires the value of m(+1, 0, -1) so as to specify the trigonometric, linear, of hyperbolic mode. The user specifies whether the Z or Y register sign is to be tested during the algorithm. The program prints out the final values of X, Y, Z, K_m and K_m^{-1}.

The initial values, X_o and Y_o, are input by the user.

Z_o must be specified in degrees for the trigonometric and hyperbolic modes. It is also possible to print out the intermediate values of the registers in decimal, both decimal and binary, or to print the final register values directly. N, the precision of the computation, can also be specified. Figure 3.1 shows the format of the program printout.

The final binary result is then converted back to decimal and the relative error is computed by using the Call-OS FORTRAN double precision routines:

$$\text{Error in \%} = 100 \times \frac{\text{CORDIC Value - True Value}}{\text{True Value}}$$



Each box represents a number (decimal or binary), result of step n.

Figure 3.1 Format of the Simulation Program Printout

3.2 Trigonometric Mode

Table A.1 of the Appendix lists the output for the trigonometric mode. It shows:

-- The initialization

-- The first 90 degree rotation

-- First step

-- Second step

-- 19th step

of an iteration in the trigonometric mode. The 19th iteration will not change the outputs X, Y, Z. In the formulas

$$X_{i+1} = X_i + 2^{-i} Y_i$$

and

$$Y_{i+1} = Y_i - 2^{-i} X_i,$$

the additive terms $2^{-i} Y_i$ and $2^{-i} Y_i$ will not contribute to the final 20-bit-resolution number for i greater than 18. The first two steps are the 90 degree rotation and 0 shift.

For the example chosen, the initial vector (X_o, Y_o) has the value $(1/K_1, -1/K_1)$ so that the final result should be:

$$X \to K_1 \sqrt{(\frac{1}{K_1})^2 + (-\frac{1}{K_1})^2} = \sqrt{2}$$

$$Y \to 0$$

$$Z \to Z_o + \tan^{-1} (\frac{-\frac{1}{K_1}}{\frac{1}{K_1}}) = Z_o - 45°$$

Figure 3.2 shows the first steps of the algorithm.

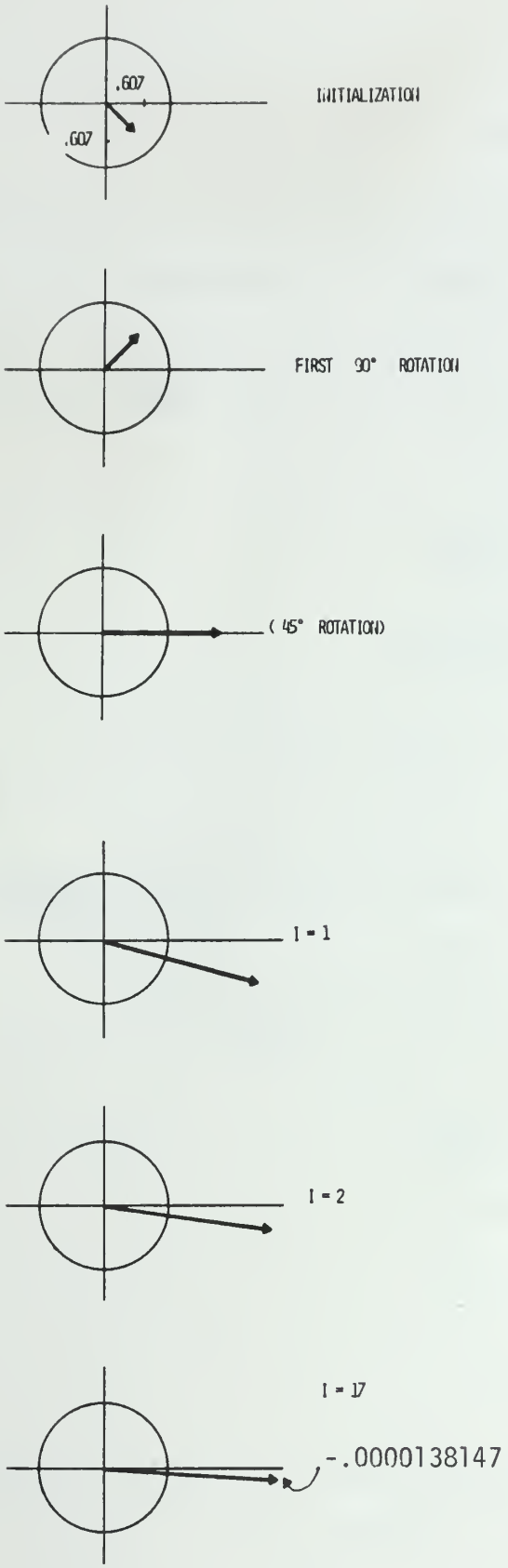


Figure 3.2 Steps in the Trigonometric Mode

3.3 Linear Mode

The example of division is used with constants equal to 2^{-1}. Figure 3.3 is a geometric interpretation of the algorithm with $Y_o = -.3$ and $X_o = .6$. The result of the division is -.4999961853.

Tabulation for the linear mode is given in Table A.2 of the Appendix.

3.4 Hyperbolic Mode

The equations,

$$X \rightarrow K_2 (X_o \cosh Z_o + Y_o \sinh Z_o)$$

$$Y \rightarrow K_2 (Y_o \cosh Z_o + X_o \sinh Z_o)$$

$$Z \rightarrow 0$$

are tabulated in Table A. 3 in the Appendix.

The value of K_2 is 0.82978162013 with $X_o = 1/K_2$ and $Y_o = 0$ and the result is:

$$X = \cosh Z_o$$

$$Y = \sinh Z_o$$

$$Z = 0$$

Iterations 4 and 13 have been repeated twice to insure convergence as shown in a previous section. One can see that the precision of these algorithms is not uniform. The simulations were done with no

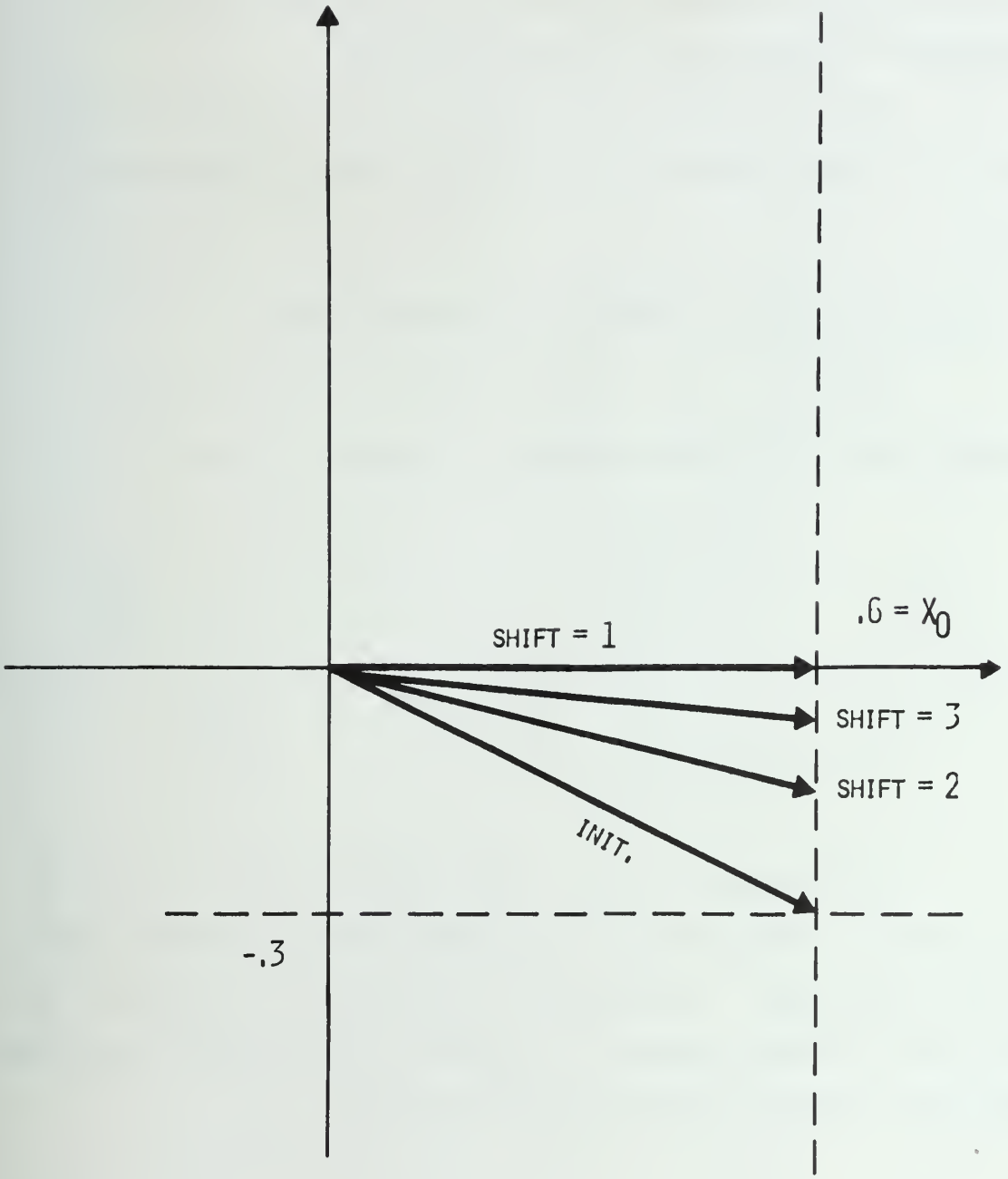


Figure 3.3 Geometric Interpretation for Linear Mode

initial normalization and thus the relative precision has some variations which are a function of the initial values, X_0, Y_0 and Z_0. In analyzing the total precision, one must account for the fact that values are represented in binary with finite length registers.

The next section gives an indication of the behavior of the error for a typical computation of the slant range at various bearing angles.

3.5 Precision of Results without Initial Normalization

Figure 3.4 represents the relative error for the X and Z registers when the following formulas are computed in binary:

$$X \rightarrow K_1 \sqrt{X_0^2 + Y_0^2}$$

$$Y \rightarrow 0$$

$$Z \rightarrow Z_0 + \tan^{-1} \frac{Y_0}{X_0}$$

Several angles ranging from 90^o have been chosen. X_0 and Y_0 have values of $\lambda \cos A_0$ and $\lambda \sin A_0$; A_0 being the chosen bearing angle and λ the slant range. The percent error is plotted as a function of the slant range for various bearings. The curves correspond to 20 significant digits. The error is computed in double precision.

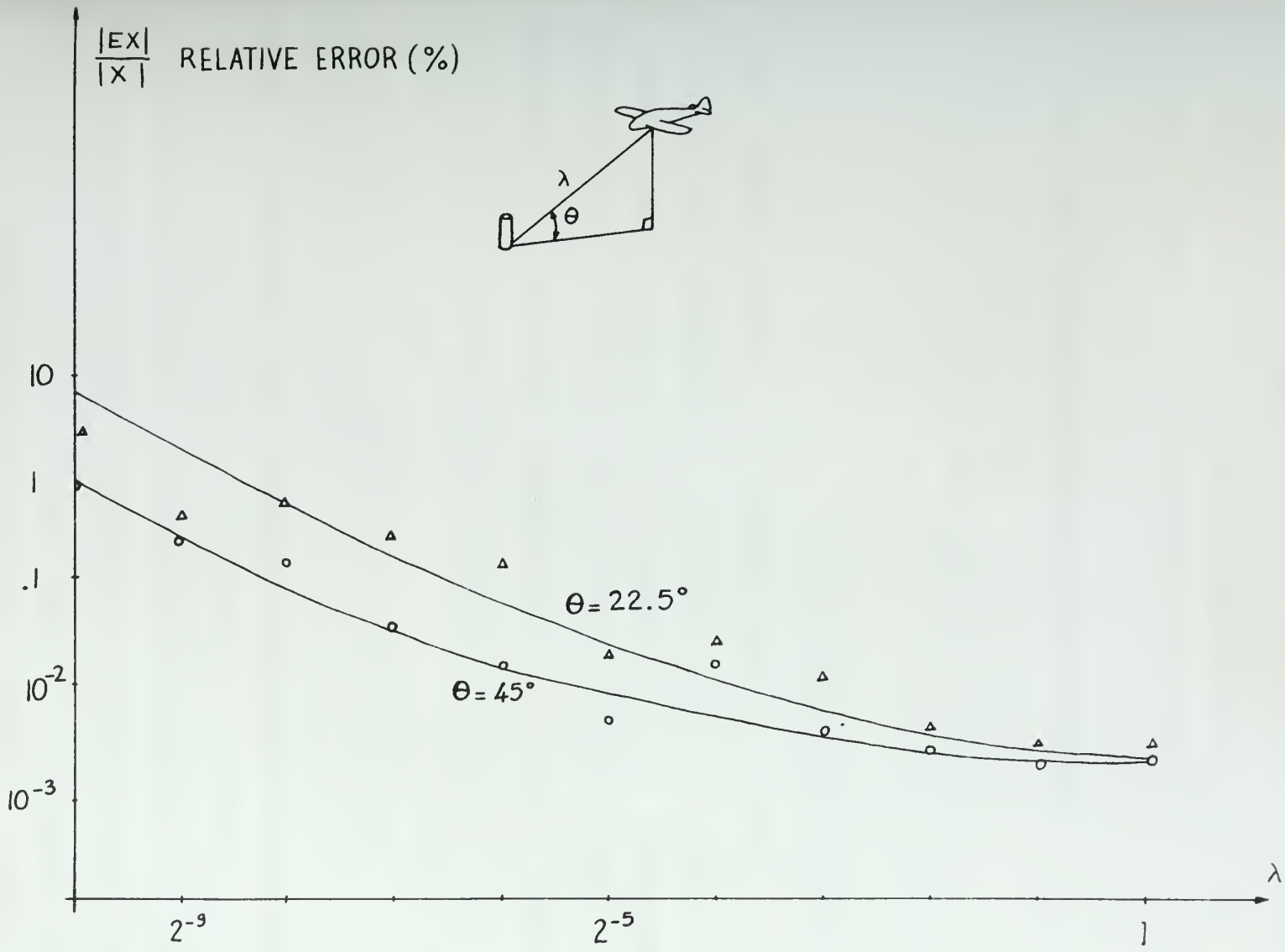


Figure 3.4 Relative Error as a Function of the Slant Range λ

4. THE 8-BIT PROTOTYPE OF A CORDIC ARITHMETIC UNIT

4.1 Hardware Realization

The prototype, constructed with T^2L technology is composed of:

i) a microprogrammed control unit

ii) the X, Y, and Z sections of the arithmetic unit including the 8-bit scalers. The use of these particular 8-bit position scalers presented some problems in the 2's-complement notation. These are explored in Section 4.3.

In Section 4.4 the design and operation of the prototype is discussed in detail.

4.2 Control of the Arithmetic Unit

The term "microprogramming" was introduced two decades ago by M. V. Wilkes. His intent was to offer a more systematic approach to control design in digital computers. Within the last few years greater understanding of the digital process combined with improved manufacturing technology rendering highspeed changeable control economically feasible has led to a resurgence of interest in computers whose controls may be modified during use.

The working nucleus of a conventional digital computer is the central processing unit or CPU and its associated main storage unit. Data flows from several parts of the computer to and from the CPU and/or memory so that operations can be performed on it (addition/subtraction and other possible elementary operations). During the basic time slot or cycle an instruction is performed and some gates are opened, counters are incremented, and so on by control signals. Depending on the present state, the machine will execute another instruction (next state).

In the conceptually simplest possible organization (Figure 4.1), the storage part of the control section would contain a storage element corresponding to each control line connected directly to its control terminal. This usually requires too many control lines (100 is a common number) and the usual procedure is to examine the control signals in groups that are logically mutually exclusive to reduce the length of the microword.

It is also necessary to establish the microprogram address (control state) that is to succeed the current one (Figure 4.1). A common technique is to construct the next address from the current address by providing in the microword format a field that controls the modification of the current address as a function of the current state. This provides the possibility of conditional jumps and GOTO-like micro-instructions.

Because of the decreasing cost of ROMS and the flexibility of microprogramming these techniques are becoming more and more popular for the control design of computers. Studies have shown that above a certain size the cost of microprogramming is less expensive than the cost of control by standard techniques.

Mode of Operation of the Control

The microprogram will consist of a series of micro-routines, one for each different machine instruction defined by the user, and a master macro-routine to effect branching to the appropriate routine for the current instruction. Each instruction in the machine program can be thought of as a macro-call with an operation code making the call, and the remainder of the instruction supplying the parameters.

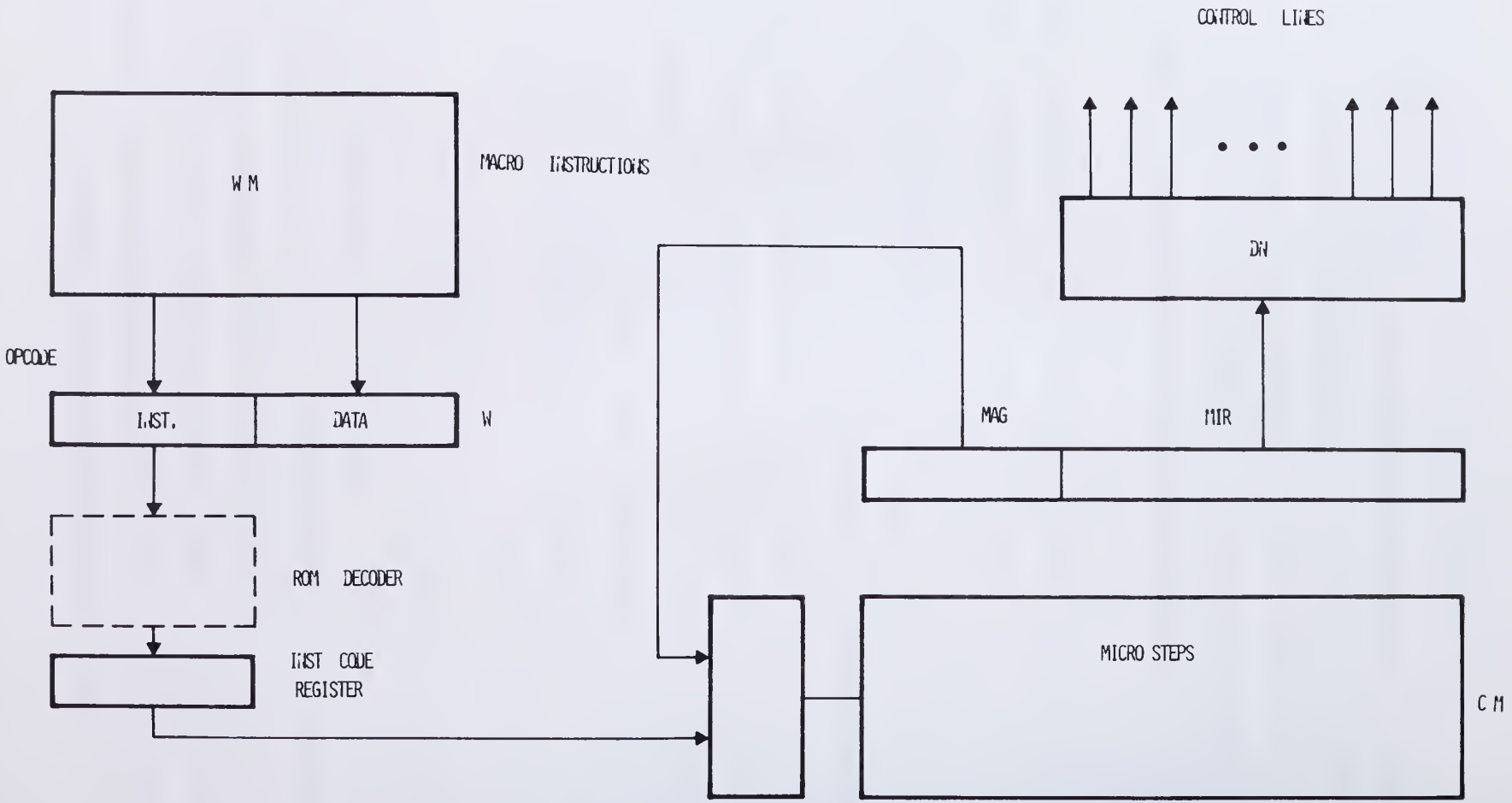


Figure 4.1 General Microprogramming Unit

There is a need for a ROM decoding table which has as many entries as there are possible values of the operation code for each processor. It is also necessary to have a way of doing a TEST and BRANCH operation as well as a RETURN from a subroutine.

Structure of the Basic Microprogrammable Unit

Although a very general microprogrammable unit has been described its implementation is simple and requires only a small amount of hardware. BRANCH or CONDITIONAL JUMP instructions are performed to implement any algorithm represented as a flow table. A small microprogram control allowing the equivalent of a GO TO instruction, or a JUMP TO SUBROUTINE (conditionally or unconditionally), will be implemented. This likewise uses a small amount of hardware (Figure 4.2).

The sequence of instructions to be executed is implemented as a flow chart rather than as a detailed microprogram subject to modifications. Thus, the microprogram part can be easily implemented during the microprogramming phase when all the control lines are known and well defined.

The organization of the control card is shown in Figure 4.2. It is comprised of: a RAM or a ROM (with no more than 16 words) where the microsteps are stored and a micro-instruction address register which has four different fields - the test, true address, false address and the micro-instruction. Many other configurations are possible, of course.

The test field represents a three bit code yielding one of eight possible tests. The tests are selected by an 8 x 1 multiplexer addressed by the test field. A memory location can be selected from among four possible sources: the true link, the false link, the stack, and a set of

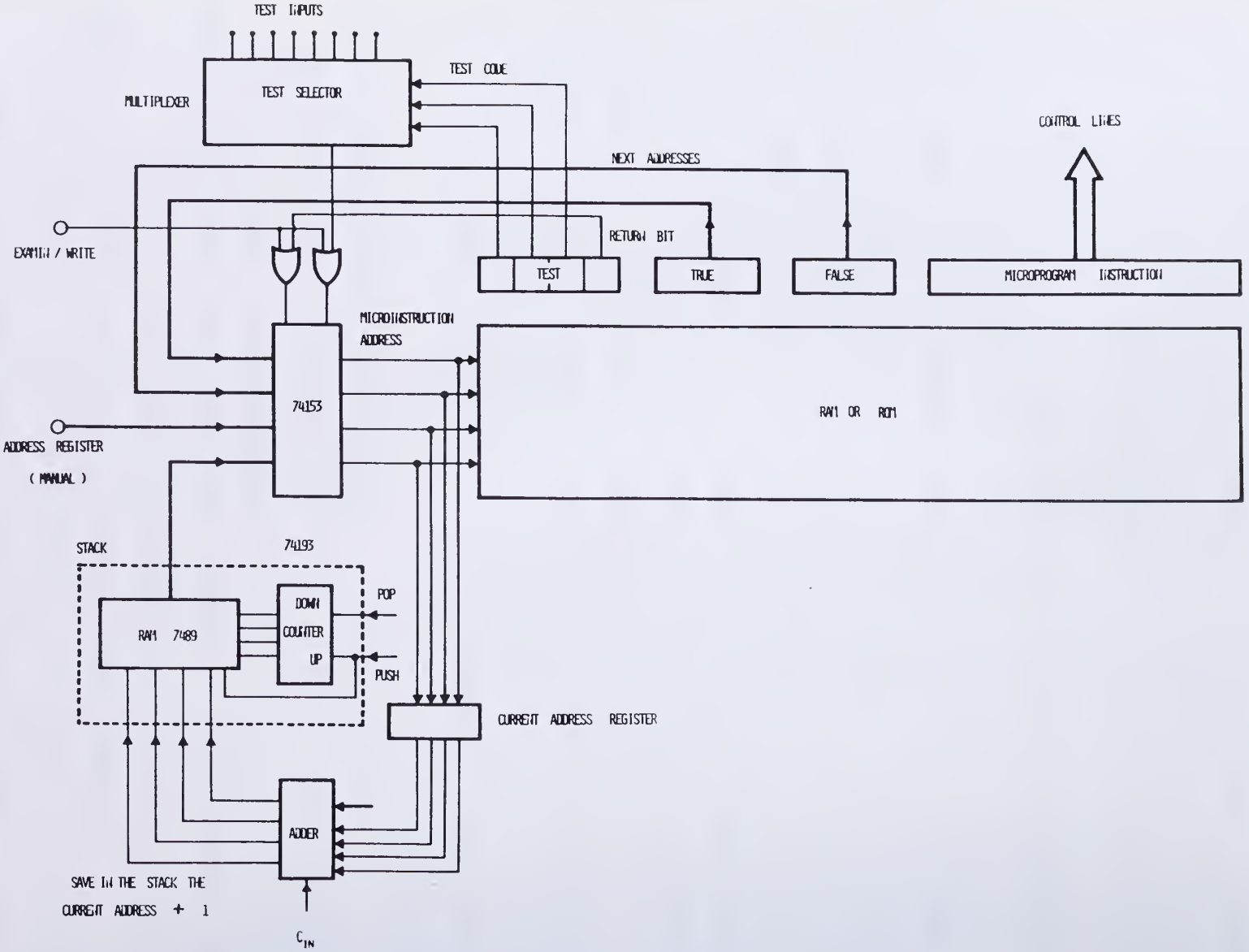


Figure 4.2 General Microprogrammed Control

switches for manual addressing (to examine the contents of a given memory location). The stack is used to save the value of the current-address-plus-one whenever the program jumps to a subroutine. The current address in incremented by one by means of an adder. To simulate a stack, a RAM and an UP-DOWN counter are used. The two control lines, "Push" and "Pop", are part of the micro-instruction. This configuration allows asynchronous control of the input/output flow between the machine and the computer.

The details of the clock sequences are shown in Figure 4.3 for a positive or negative edge triggered control line. The delay, , will be necessary to allow the control bits A and B to be latched into the micro-instruction register before the occurrence of the clock edge if edge triggered devices are used. The JUMP TO SUBROUTINE will issue a signal on the PUSH control line while the POP is 0 (Figure 4.3). The return signal will POP the preceding address from the stack and cause the data selector switch to get the next address from the stack.

4.3 The Use of Scalers for 2's Complement Shifting

The CORDIC algorithms require shifting by positions varying from 1 to n for a typical n-step algorithm. However an eight-bit position scaler which performs only end-off shifts was used. Figure 4.4 shows the diagram and truth table of the scaler. This basic 8-bit building block can be used for any shift function.

It is possible to use a simple shift register which shifts any number of places right or left. However, an 8-bit position scaler package takes no more space than an 8-bit parallel-in, parallel-out shift register. In addition, the scaler is faster and has simpler control.

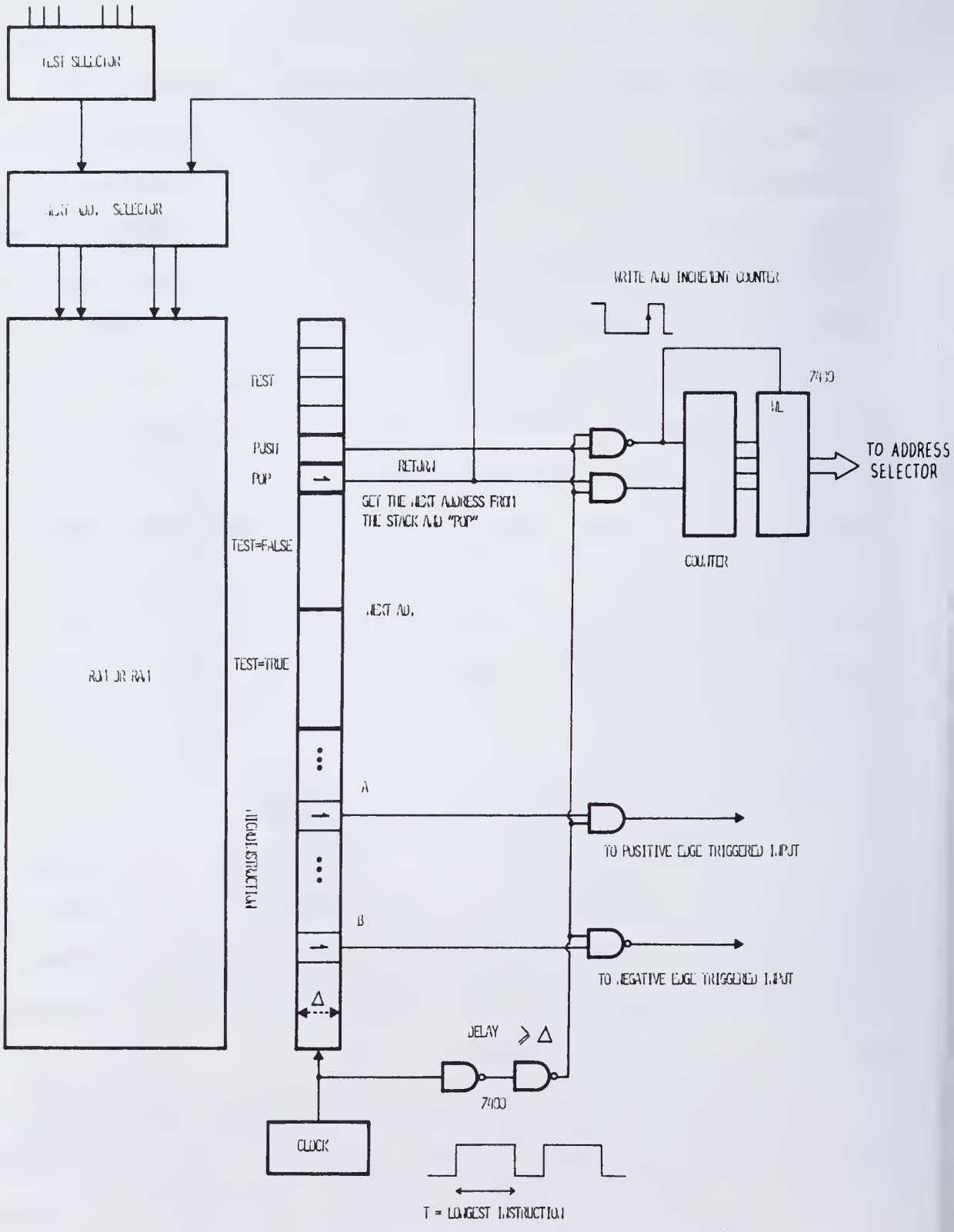
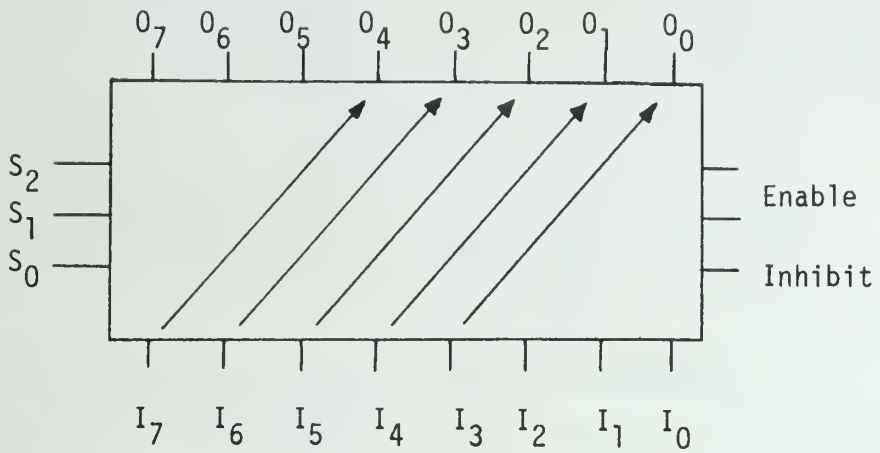


Figure 4.3 Microprogram Control and Timing



Example with a shift of 3 ($S_0S_1S_2$ = 110)

Truth table :

INHIBIT	ENABLE 1 & 2	S_0	S_1	S_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	1	0	0	0	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$
0	1	1	0	0	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	1
0	1	0	1	0	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	1	1
0	1	1	1	0	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	1	1	1
0	1	0	0	1	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	1	1	1	1
0	1	1	0	1	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	1	1	1	1	1
0	1	0	1	1	$\overline{I}_6$	$\overline{I}_7$	1	1	1	1	1	1
0	1	1	1	1	$\overline{I}_7$	1	1	1	1	1	1	1
1	x	x	x	x	1	1	1	1	1	1	1	1
x	0	x	x	x	1	1	1	1	1	1	1	1

Figure 4.4 Scaler and Truth Table.

Scalers belong to the more general class of uniform shift networks. This includes

. Scalers or shift circuits for end-off shifting

. Rotate circuits for end-around shifting

. Shift and rotate circuits or "barrel switches" which are capable of both end-off and end-around operation.

The outputs of the scaler used are open collector to allow for array expansion.

However, for 2's complement numbers, there is the problem of sign propagation for right-shifting. A number in 2's complement form with the left most bit being the sign, must have its left most bits filled in with the sign bit when divided by 2^n (Figure 4.5).

This scaler replaces the left most bits by 1's independently of the input binary number. Some kind of connection is therefore needed in order to insure proper sign propagation.

The number of units required for an n-bit scaler is $(\frac{n}{8})^2$ (Figure 4.5)

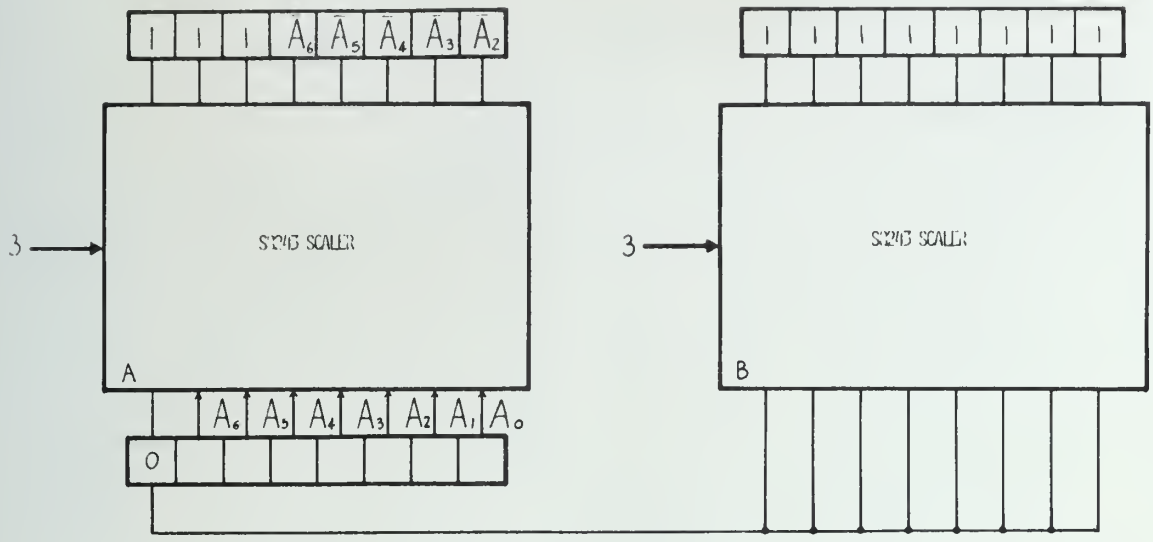
One approach implementable for small scalers, is to use $(\frac{n}{8})^2$ extra 8-bit scalers having the sign bit of the shifted number as inputs. The various cases are shown in Figure 4.5.

For all possible cases it is easy to see that if the outputs of the B scaler are complemented when the sign bit is 1 and unchanged when the sign bit is 0, and if these outputs are wire ANDed with those of the A scaler, one will have the required correction (Figure 4.6).

However, this solution is not very economical for large n.

Another approach makes use of a 32 x 8 T^2L ROM. This solution avoids the duplication of $(\frac{n}{8})^2$ barrel shifters whose only function is to propagate the sign-bit (Figure 4.7). When the sign bit is 0 the ROM sees

Positive 2's complement number:



Negative 2's complement number:

0 0 0 $\bar{A}_6$ $\bar{A}_5$ $\bar{A}_4$ $\bar{A}_3$ $\bar{A}_2$

Correction needed in this case

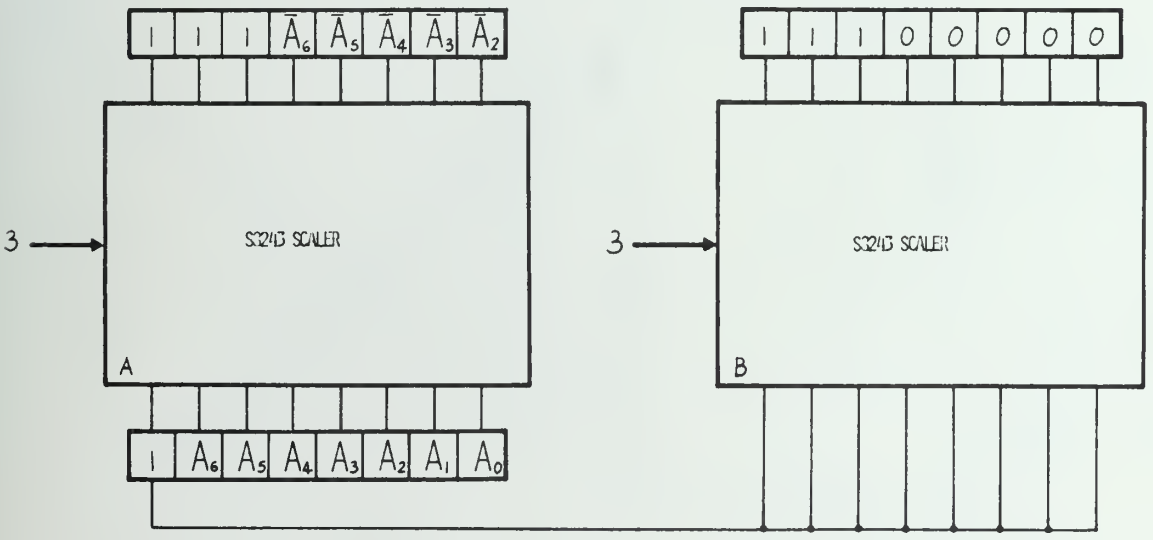


Figure 4.5 Sign Propagation with the 8-Bit Scaler

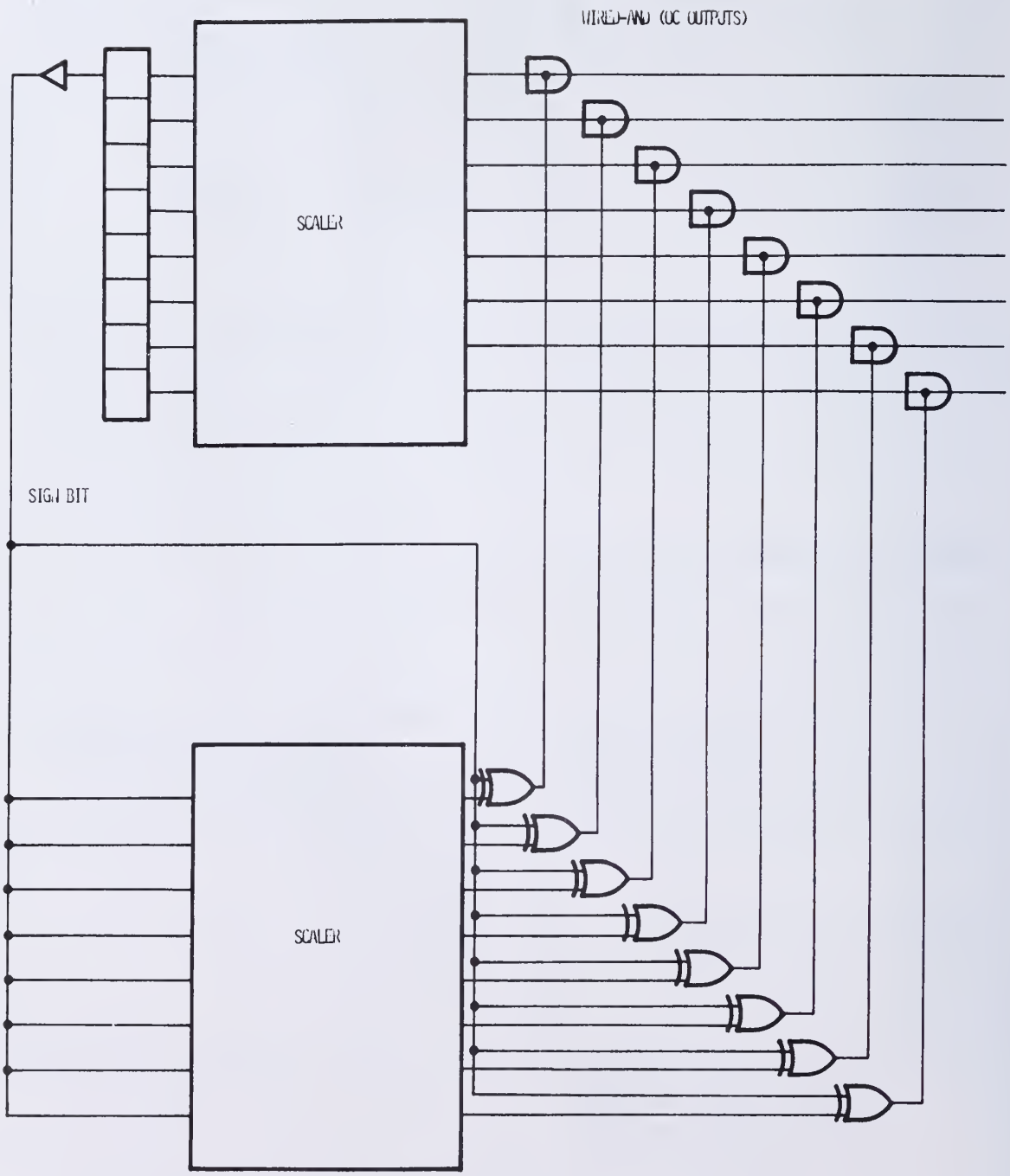


Figure 4.6 Correction of Sign Bit, First Method

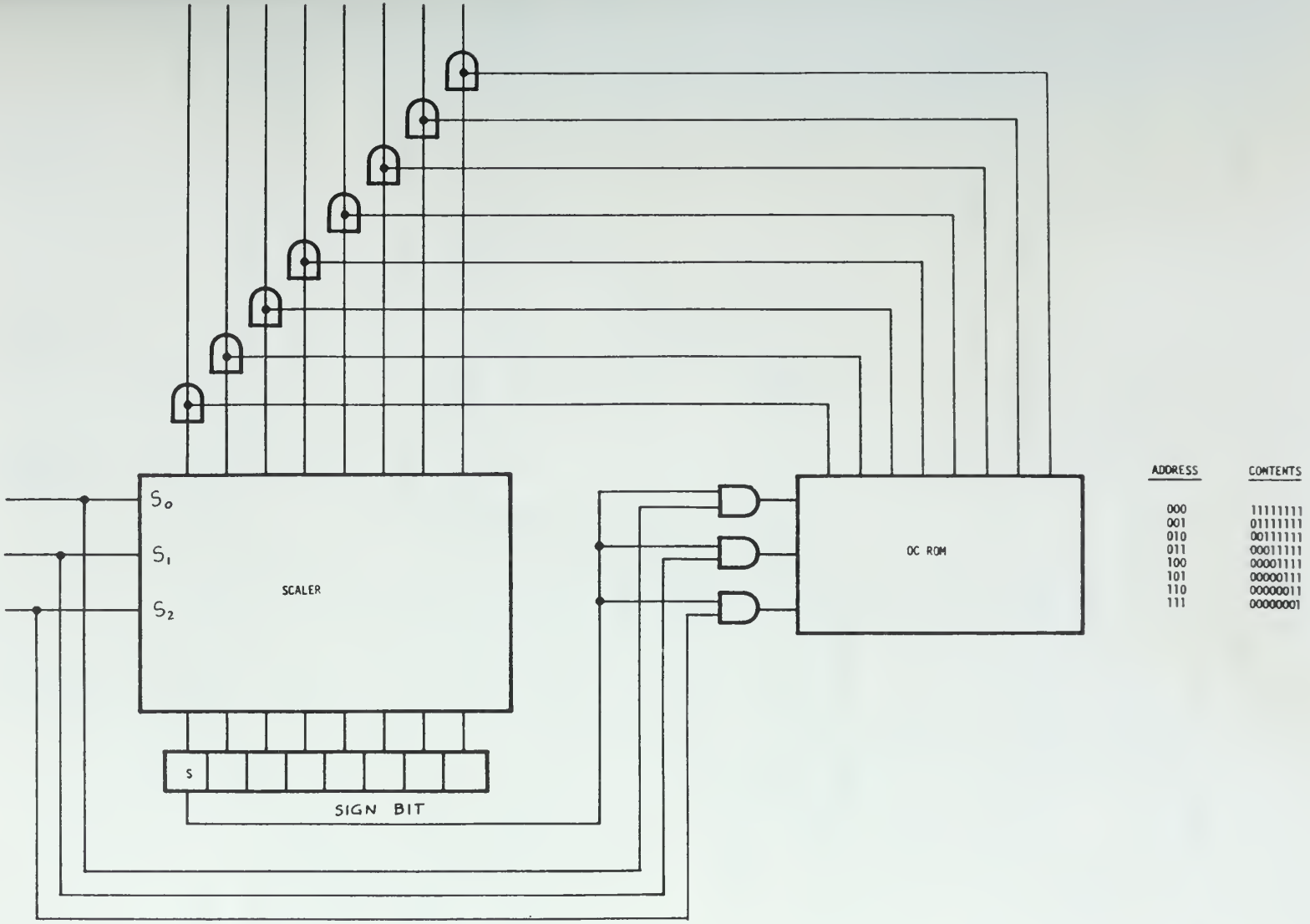


Figure 4.7 Correction of Sign Bit, Second Method

the address 000 which is loaded with 1's only. When the sign bit is a 1 the output of the ROM address is of the form: 00 .. 0111, where the number of zeros is equal to the value of the binary address.
For a 32 position scaler one would need four ROMs plus two AND-gates for the address. This is a substantially smaller number of packages than required by the previous solution. The access time of the currently available T^2L ROMS is one the order of 30 to 60 ns; compatible with the delay of the scalers.

4.4 An 8-bit Prototype CORDIC

The organization is sketched in Figure 4.8.

We can distinguish:

- The input mutliplexers
- The CORDIC arithmetic unit
- The output registers

The cross connection implementing the "cross addition/substraction" is shown. The X, Y and A registers can be initialized independently by means of switches. The counters are used to address the scalers and the ROM containing the constants.
The microprogram is stored in a RAM: A set of switches allows debugging and changing the micro instructions in order to implement any of several algorithms. The following labels have been chosen:

- Input multiplexers: MPY, MPX, MPA
- Counters: CTSH (Count/Shift, for addressing the scalers)
  CTA (for addressing the ROM storing the ATR constants)
- Input registers: IRY, IRX, IRA
- ROM storing the constants: ROMA

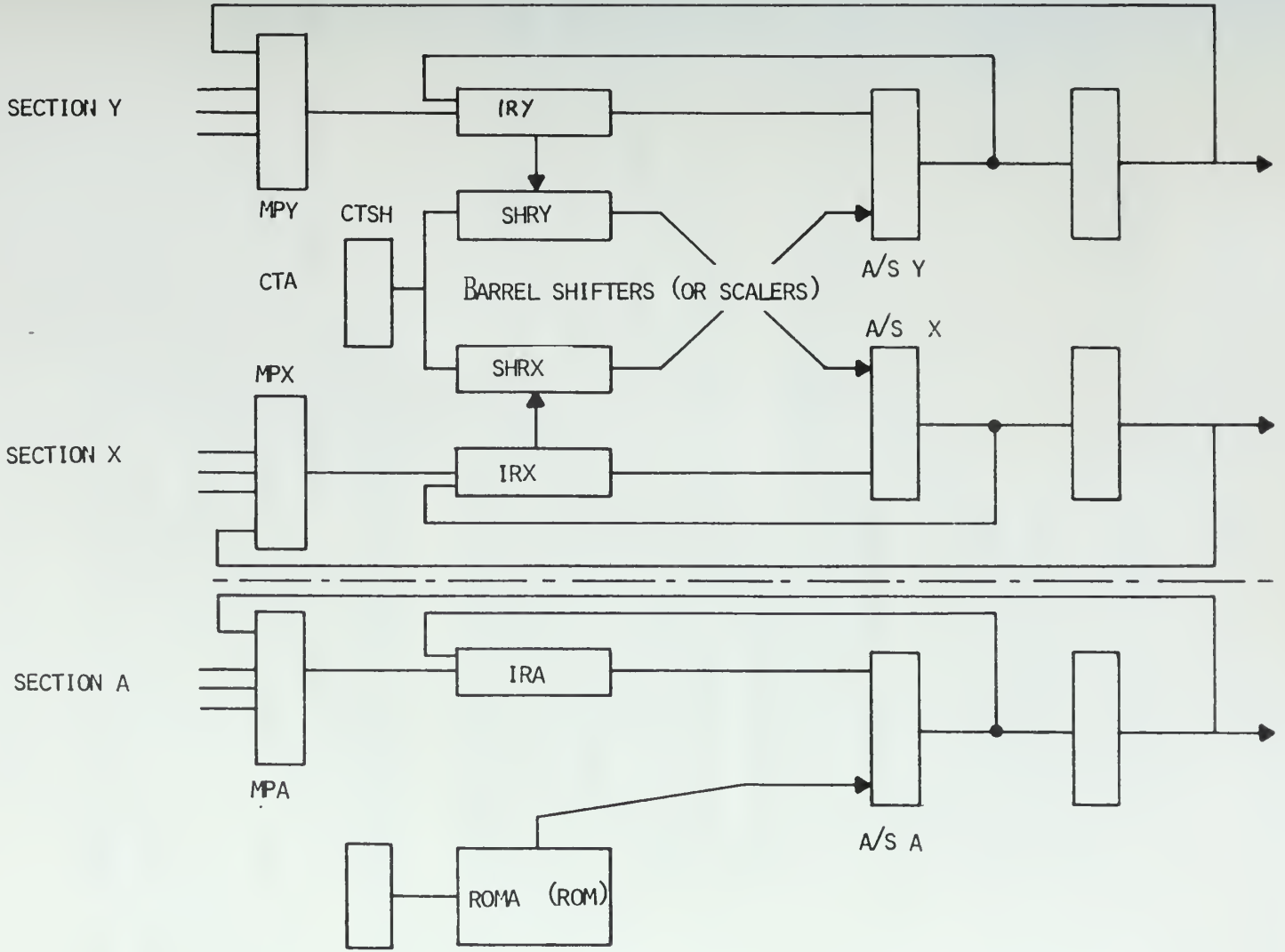


Figure 4.8 Block Diagram of the CORDIC Unit

- The adder-substracters: A/SY, A/SX, A/SA
- The output registers: ORY, ORX, ORA

Microinstruction Format

I: Input gating signals

O: Output gating signals

CORDIC: A/S, Counters, and ROM

NEXT ADDRESS: The binary value of the jump address or next address. This format is convenient and flexible for this prototype of the AU.

Figure 4.9 shows the diagram of the control.

The execute signal opens gates and takes into account the delay in performing an instruction. The fetch signal latches the new address into the instruction address register and issues the next instruction to the input of the microinstruction register.

Control-Microinstruction Address Register

Figure 4.10 shows the microinstruction address register. The dual 4-line to 1 line Data Selector/Multiplexers allow the selection of the address for the RAM either from manual switches or from the next-address portion of the microinstruction. The singleshot is used to generate the fetch signal, which can be initiated manually also.

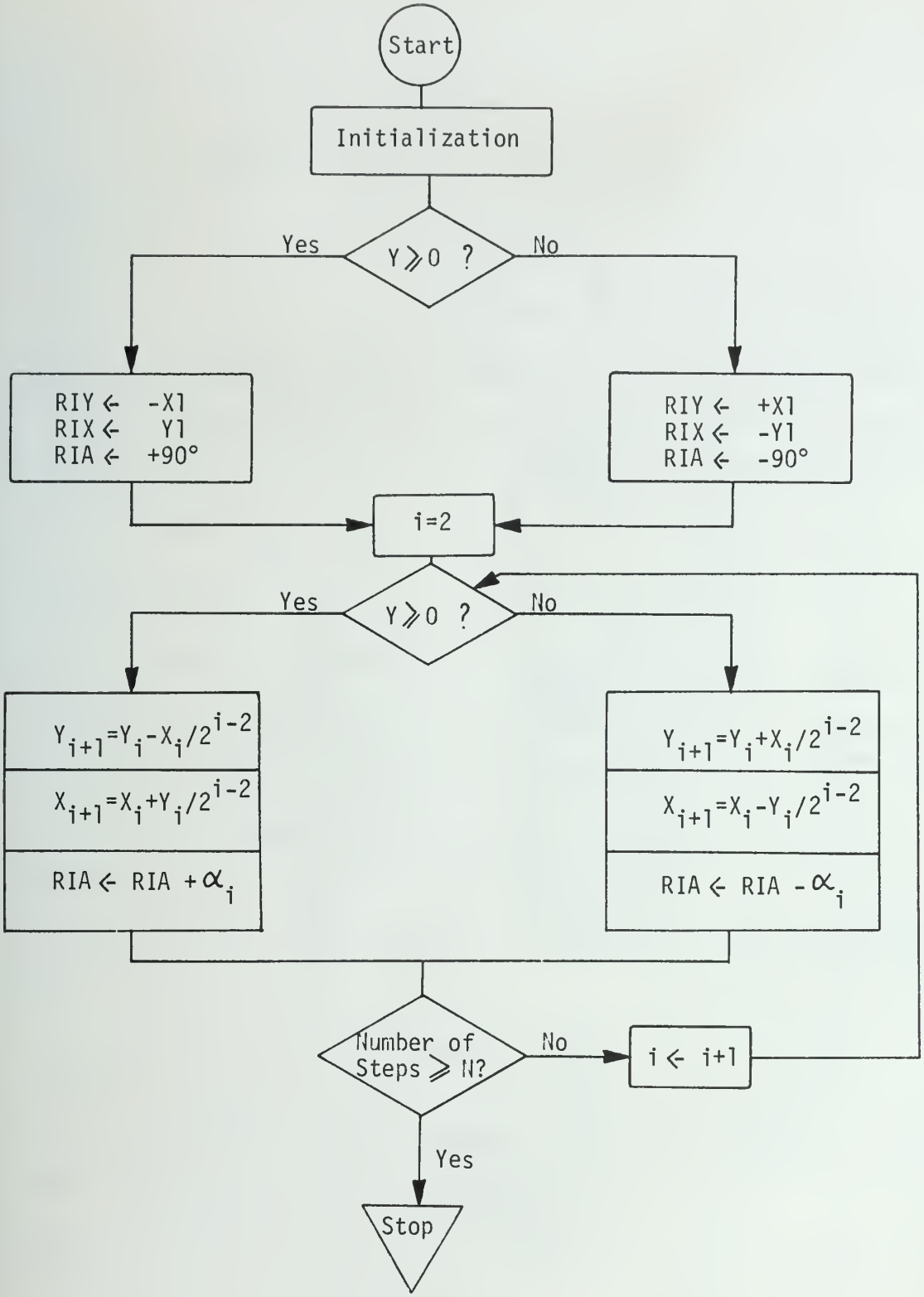


Fig. 4.9 Control Flowchart (Trigonometric Mode)

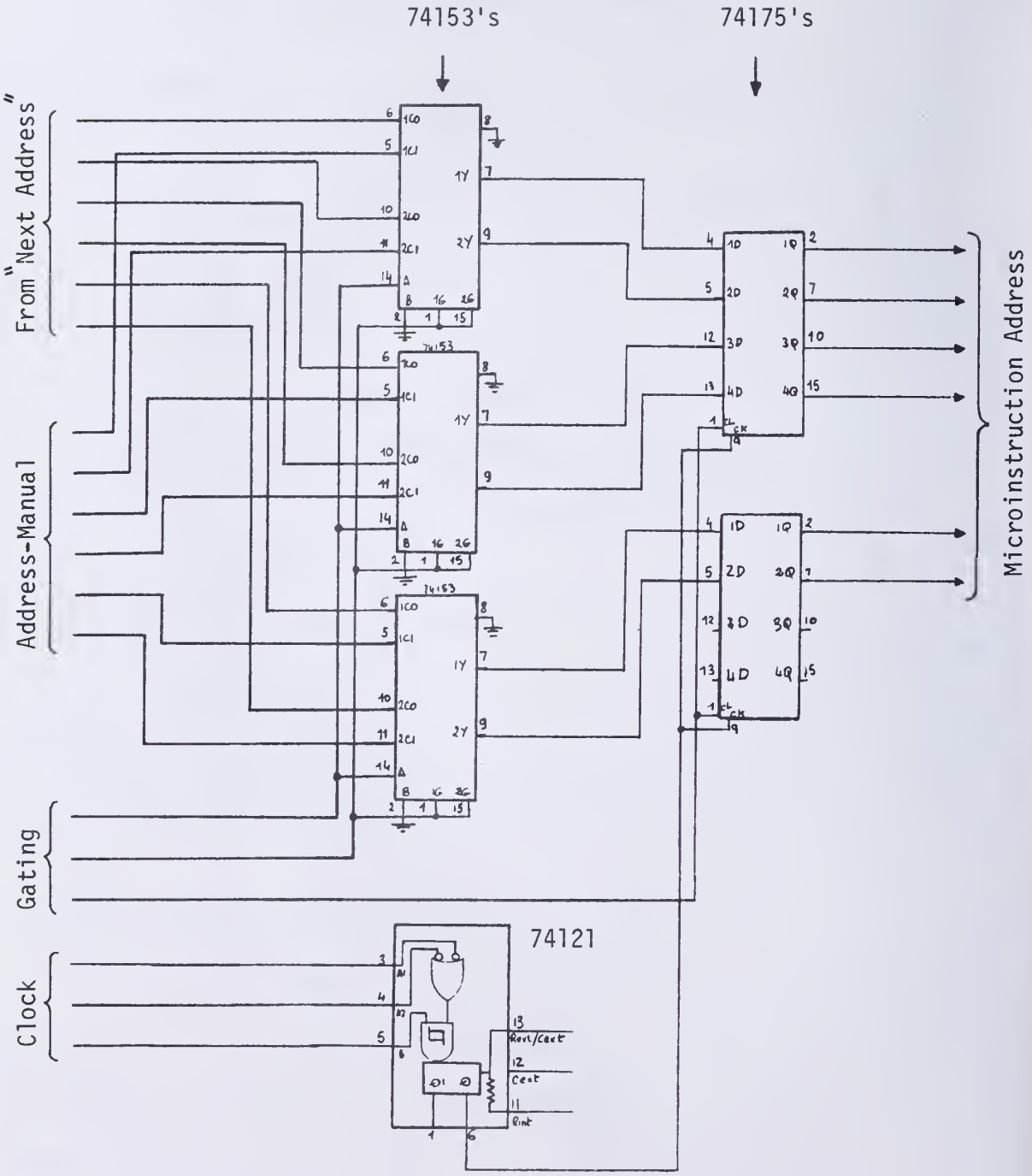


Figure 4.10 Microinstruction Address Register

Control-Microinstruction Register

Figure 4.11 shows the microinstruction register. It is an edge-trigerred latch since some input signals return to their initial value and some remain at a new value.

In other words, some control signals are pulses and some are constant. The execute pulse is delayed and used to generate the pulse signals whereas the level signals come directly from the MIR. The single shot generates the "execute" pulse. The inputs to the MIR come from a 32 x 20 bit RAM.

Input Multiplexers

Refer to Figure 4.12. Four mutliplexers are used to steer the values X_n, Y_n, and A_n or to initialize the registers X, Y and A depending on the mode. Switches are used for initialization. The select inputs are labelled A and B; the strobe, G; the data inputs C0, C1, C2, C3 and the outputs Y. The strobe, G, can be used to clear RIY, RIX and RIA, since, if G is a 1, the output of Y is low no matter what the inputs may be.

Outputs

Figure 4.13 shows the three 8-bit output registers and the strobe signals GOX, GOY, and GOA.

CORDIC

Figures 4.14, 4.15, and 4.16 show the CORDIC unit itself. One can distinguish

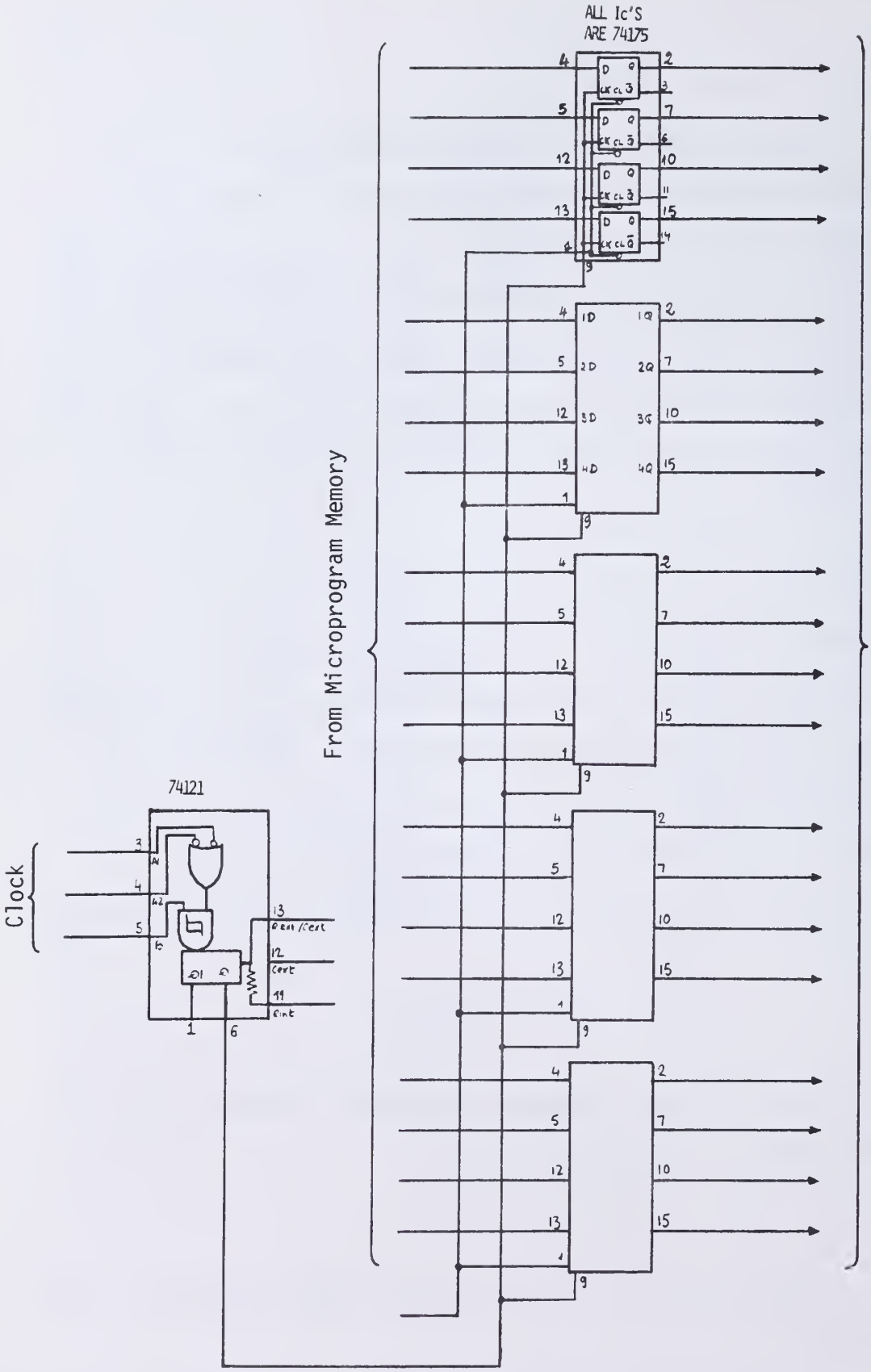


Figure 4.11 Microinstruction Register

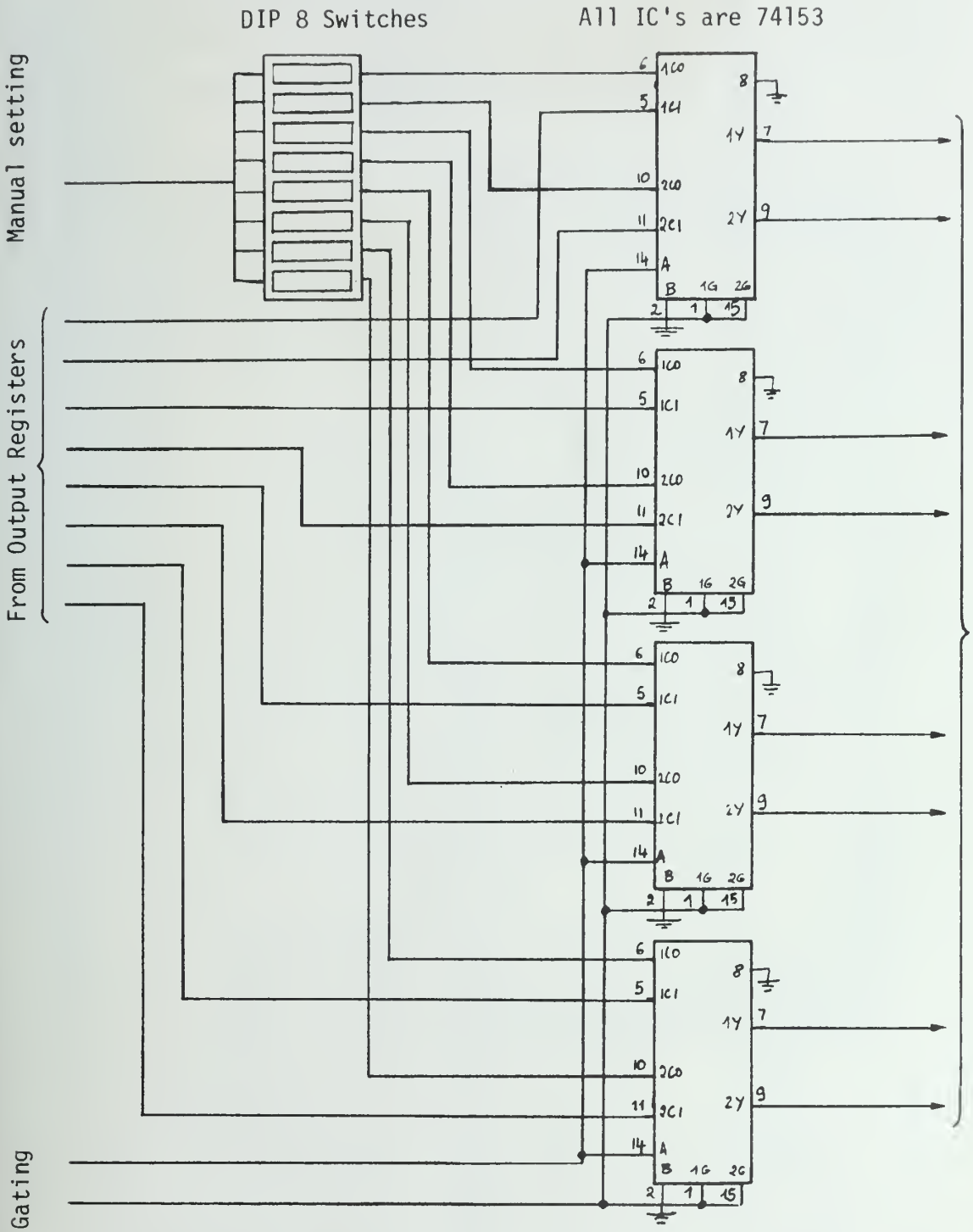


Figure 4.12 Input Multiplexers

All IC's are

7475

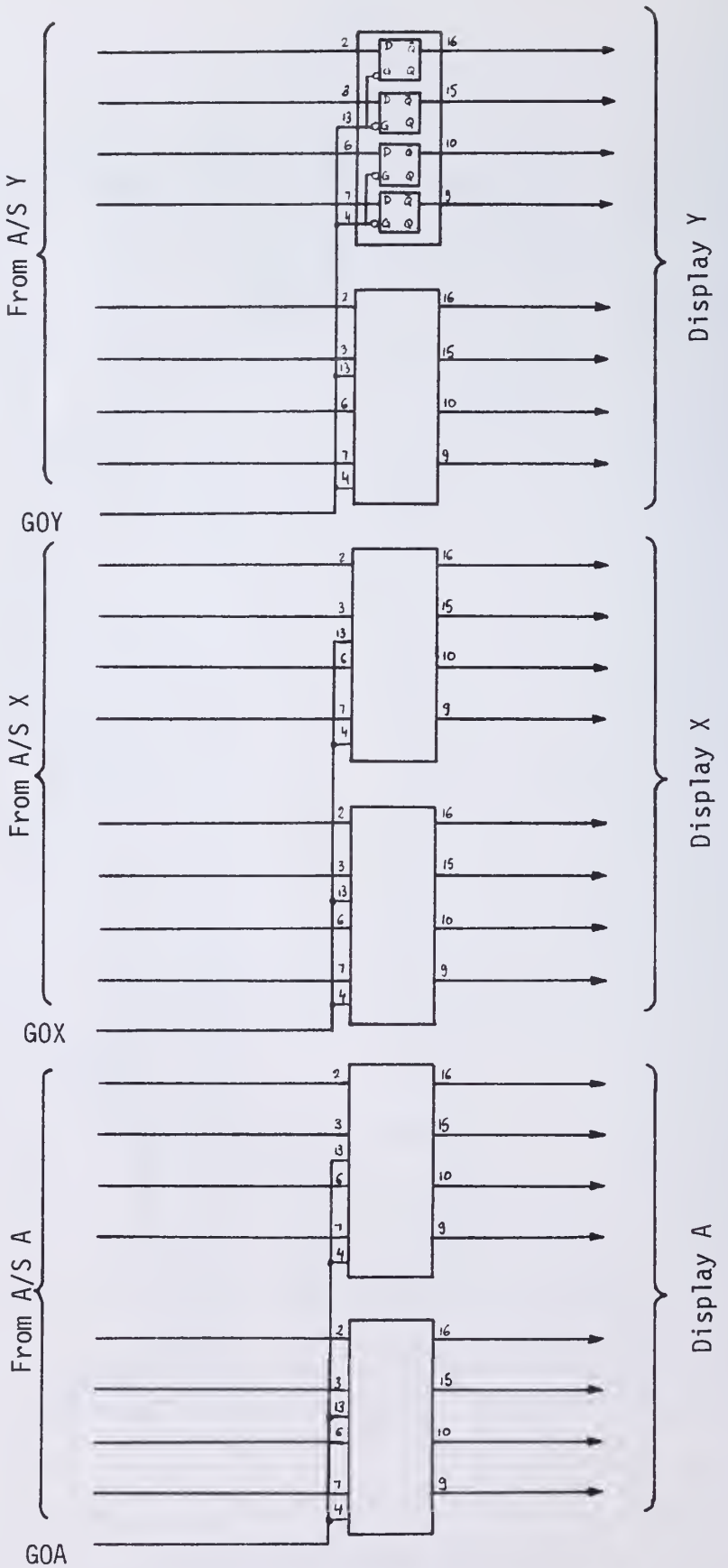


Figure 4.13 Outputs

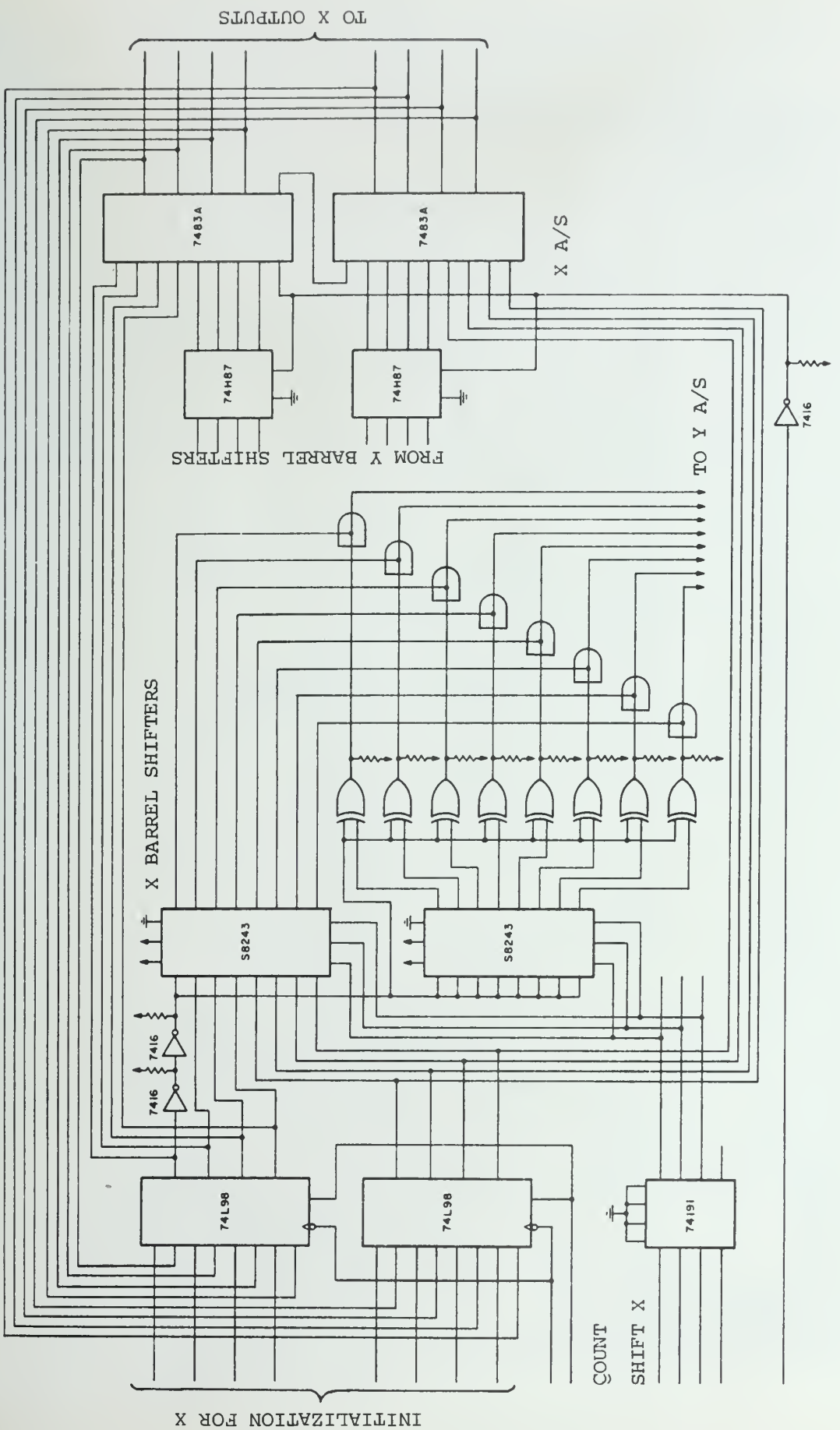


Figure 4.14 CORDIC Arithmetic Unit, X Portion

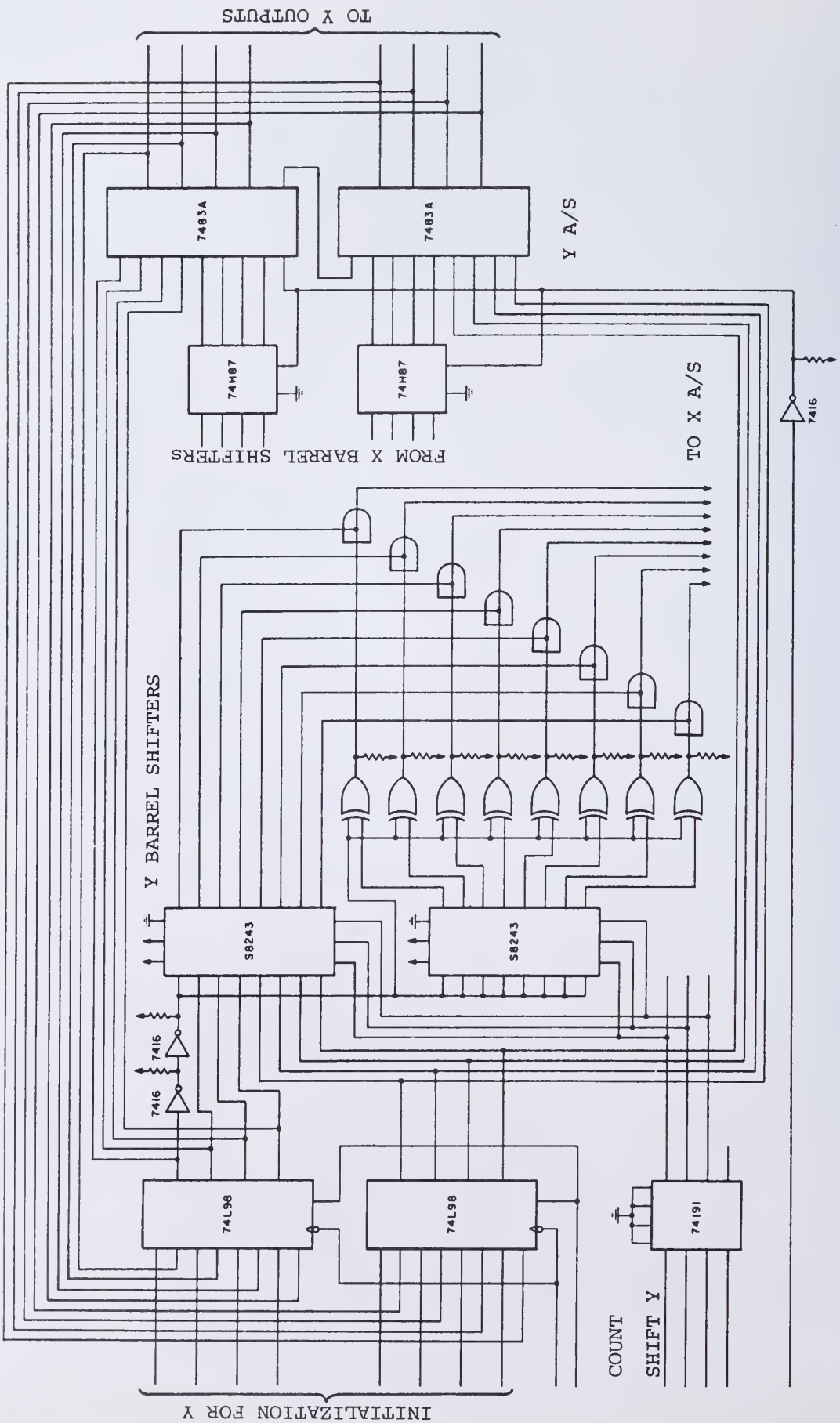


Figure 4.15 CORDIC Arithmetic Unit, Y Portion

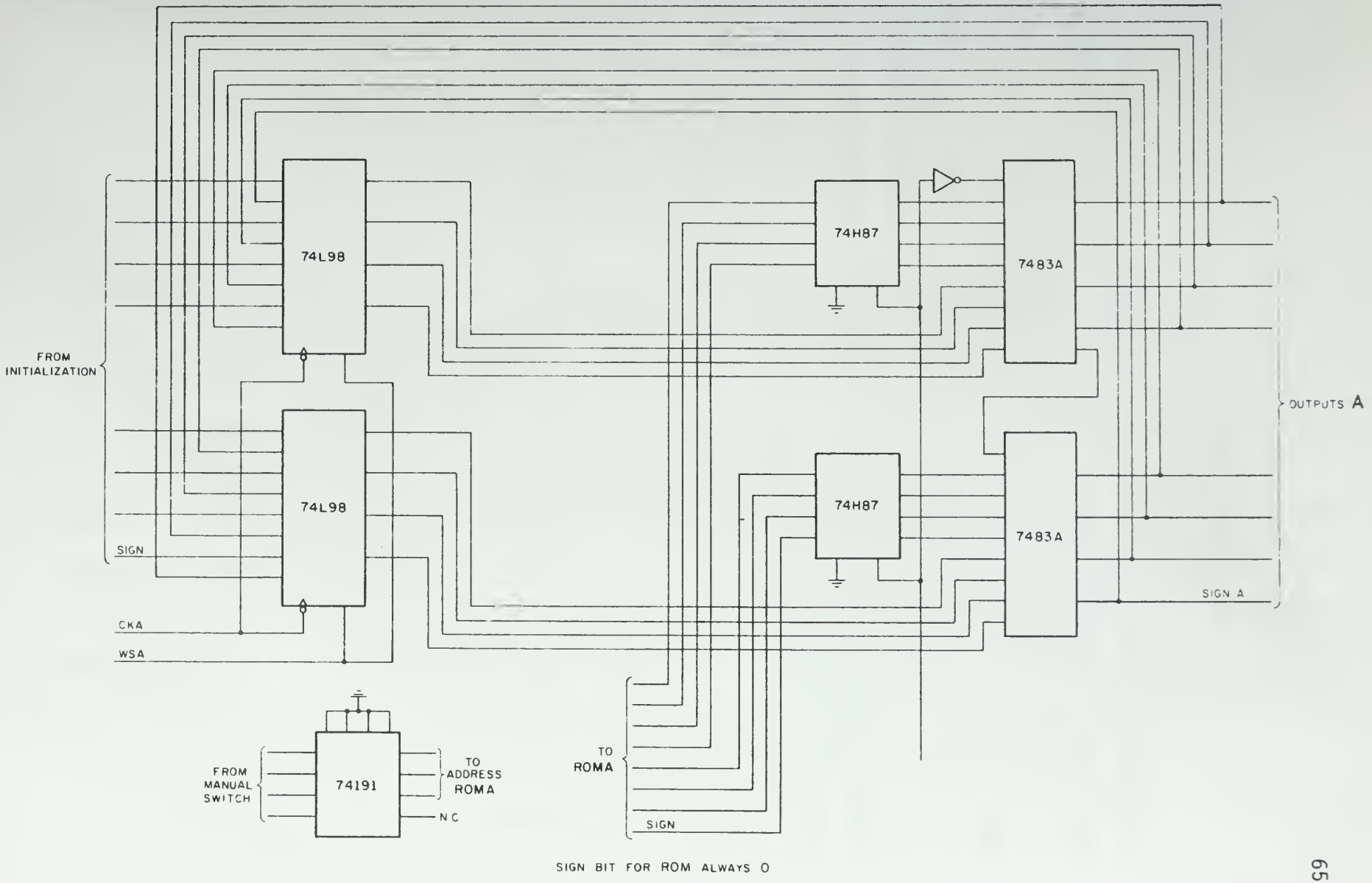


Figure 4.16 CORDIC Arithmetic Unit, A Portion

. the input register stage,

. the barrel switch stage, and

. the adder/subtracter stage.

In Figure 4.16,the "angle" portion of the arithmetic unit, shows that the inputs to the adder/substracter are taken directly from the ROM, the scalers being unnecessary for the A portion of CORDIC.

Speed

The microprogram is shown in Table 4.1 and requires ten steps. Due to the limitation of the stored constant ROM used, the minimum step duration obtained is $\frac{1}{1.2\ \text{Mhz}} = 850$ ns. The ten steps required by the algorithms thus take 8.5 μs.

INSTRUCTION		ADDRESS	AI	CLYA	CLX	YORZ SIGN	CLEAR CNTRS	GOX	CKI	WS	CKSH	CKRMA	FREEZE GATE SIGN	GO_{AY}	NEXT ADDRESS
clear everybody	0	0000	0	1	1	0	0	1	1	0	0	0	1	1	0001
0 rotation	1	0001	0	0	1	0	1	1	1	0	0	1	1	0	0010
1st rotation	2	0010	0	1	0	0	1	0	1	0	0	0	0	1	0011
2nd rotation	3	0011	1	0	0	0	1	0	1	0	0	1	0	0	0100
3rd rotation	4	0100	0	0	0	0	1	0	1	1	0	1	1	0	0101
4th rotation	5	0101	0	0	0	0	1	0	1	1	1	1	1	0	0110
5th rotation	6	0110	0	0	0	0	1	0	1	1	1	1	1	0	0111
6th rotation	7	0111	0	0	0	0	1	0	1	1	1	1	1	0	1000
7th rotation	8	1000	0	0	0	0	1	0	1	1	1	1	1	0	1001
8th rotation	9	1001	0	0	0	0	1	0	1	1	1	1	1	0	1010
Latch results	10	1010	0	0	0	0	0	0	0	1	0	0	1	1	1010 or 0000

Table 4.1

Microprogram for the Trigonometric Mode

5. METHODS BASED ON FUNCTION APPROXIMATION

5.1 High Speed Division Using High Speed Multipliers

There exist several algorithms in the literature dealing with the computation of the reciprocal, $\frac{1}{x}$, of a normalized number, x. Wallace [WAL64] first developed an iterative division scheme by making use of a fast mutltiplier. The IBM/360 floating point execution unit also makes use of a fast multiplier to perform division. It is an iterative process where, on each iteration, a factor, R_k, multiplies both numerator and denominator so that the resultant denominator converges quadratically toward one and the resultant numerator converges quadratically toward the desired quotient:

$$\frac{N}{D} \times \frac{R_0}{R_0} \times \frac{R_1}{R_1} \times \frac{R_2}{R_2} \times \ldots \times \frac{R_n}{R_n} \rightarrow NR_0R_1 \ldots R_n = \text{Quotient}$$

Ferrari [FER71], Ling [LIN71] and Stefanelli [STE74] have also proposed various multiplicative schemes for computing the reciprocal of a number. An overview of these algorithms is given in Appendix A.4 Stefanelli describes an inversion algorithm based on the Taylor series. It makes extensive use of read-only memories and a set of dedicated, non-cascaded mutlipliers.

This technique is very fast because it is entirely parallel, but there are some inconveniences:

. the method uses dedicated multipliers to implement the division algorithm and we believe that such algorithms should be designed around an n x n multiplier that can be used for multiplication, as well as other function generation.

. the Taylor polynomial produces very poor behavior with respect to error and it is possible to use other approximating polynomials which yield better error control.

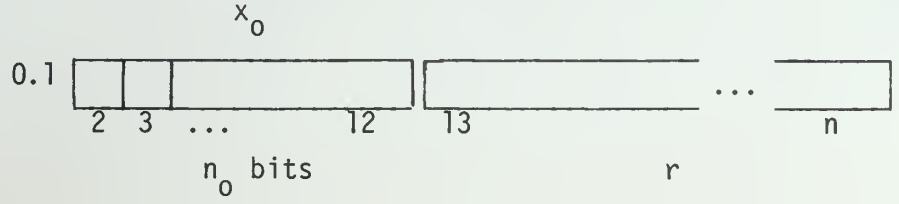
. It is expensive to use algorithms based on series approximations and ROM lookup to implement a totally parallel divider for a precision of more than 32 to 34 bits. The precision can be increased, however, by more efficient use of the ROMs than that proposed by Steffanelli [STE74] (See Appendix A.5)

As discussed later, other approximating schemes taken from numerical analysis lead to better error behavior than that obtained from a Taylor polynomial expansion.

5.2 Division and Function Generation Using Optimum First Order Polynomial Interpolation with Selected Abscissas

We now investigate the precision and the possibility of implementing function approximation by applying a technique known as optimum polynomial interpolation. The results are general and apply to any continuous function, f(x), that must be approximated between selected points x_0, x_1. The equations are easily derived for any arbitrary function f(x), although they are not optimal in the Tchebychev sense (minimax approximation). Remez' algorithm and Tchebychev polynomial approximations are two other techniques available. The following describes the notation used throughout the remaining paragraphs.

The given number x has the following format



$$x = x_0 + r \qquad 0 \leq r < 2^{-n_0}$$

$$r = R.\ 2^{-n_0}$$

$$0 \leq R < 1$$

n_0 is the number of bits looked up in a table (10 to 14 seems to be practical given the present technology).

x is the given number with n bits precision

x_0 represents the n_0 high order bits

r represents the $(n-n_0)$ low order bits

$\frac{1}{x_0}$ is stored in a ROM (rounded)

R represents the number (not necessarily normalized):

$x_1 = x_0 + 2^{-n_0}$, next interval point

f(x) is the function to be approximated ($\frac{1}{x}$, e^x, log x ...)

Optimum Polynomial Interpolation with Selected Abscissas

If a function f(x) is approximated by the interpolating polynomial $p_n(x)$ of degree n which agrees with f(x) at n + 1 points $x_0, x_1, \ldots x_n$ we have

$$f(x) - p_n(x) = \pi(x) \frac{f^{(n+1)}(\xi)}{(n+1)!} = E(x,\xi)$$

where $\pi(x) = (x-x_0)(x-x_1) \ldots (x-x_n)$

if $x_0 < x_1 < \ldots < x_n$ and $\xi \in [x_0, x_n]$

The interval must be reduced to the interval [-1, 1] by an appropriate change of variables. The term $f^{(n+1)}(\xi)$ depends on x and on the abscissas $x_0, x_1 \ldots x_n$. One can chose to minimize $|\pi(x)|$ over [-1, +1] rather than minimizing $|f(x) - p_n(x)|$. If $f^{(n+1)}(\xi)$ does not vary too much in the interval, one would obtain a nearly "best approximating" polynomial. This technique can be used to find an approximation to all functions f(x) having continuous derivatives in [-1, +1]. Thus we will minimize the quantity:

$$I = \int_{-1}^{+1} w(x) [\pi(x)]^2 \, dx$$

Where w(x) is a prescribed weighing function which is non-negative in [-1, +1]. This will minimize the error due to the factor $\pi(x)$.

It is known that

a) If w(x) = 1, the n + 1 roots of $\pi(x)$ that minimize I are the zeros of $p_{n+1}(x)$, the $(n+1)^{st}$ Legendre Polynomial.

b) If $w(x) = \frac{1}{\sqrt{1 - x^2}}$, the n + 1 abscissas are the zeros of $T_{n+1}(x)$, the (n+1) Chebyshev polynomial and are given by

$$x_i = \cos\left(\frac{2i+1}{n+1} \cdot \frac{\pi}{2}\right) \qquad i = 0, 1, \ldots n$$

It follows that $\pi(x) = 2^{-n} T_{n+1}(x)$

$$(x) = \frac{2^{n+1}\,[(n+1)!]^2}{(2n+2)!} P_{n+1}(x)$$

c) If $w(x) = (1-x)^{\alpha}(1+x)^{\beta}$ $\alpha > -1$ $\beta > -1$ the abscissas are the zeros of the Jacobi Polynomial.

Next, we will study in more detail the Legendre and Chebyschev polynomials for division and will describe a hardware implementation for various precision.

Range Transformations

The formulas used generally assume that the variable x takes values in the interval [-1, 1]. The range transformations are derived as follows: We have $x = x_0 + r$ (notation as above) with x ranging between x_0 and x_1, the mapping into [-1, +1] leads to:

$$x = \frac{2x}{x_1-x_0} - \frac{x_1+x_0}{x_1-x_0}$$

That is, $x = \frac{x(x_1-x_0)}{2} + \frac{x_1+x_0}{2}$

Legendre and Tchebychev Interpolation of First Order

The values x_0 and x_1 are the roots of $P_2(x) = \frac{1}{2}(3x^2-1)$ for a Legendre approximation of f(x) and $T_2(x) = 2x^2-1$ for the Tchebychev case.

The roots, in either case, are $x_0 = -x_1 = \alpha = \frac{\sqrt{3}}{3}$ for Legendre and $\frac{\sqrt{2}}{2}$ for Tchebychev. We have for the new abscissas:

$$x_0 = -\alpha \left(\frac{x_1 - x_0}{2}\right) + \frac{x_1 + x_0}{2} = -\alpha \frac{h}{2} + \frac{x_1 + x_0}{2}$$

$$x_1 = \alpha \left(\frac{x_1 - x_0}{2}\right) + \frac{x_1 + x_0}{2} = \alpha \frac{h}{2} + \frac{(x_1 + x_0)}{2}$$

The new first order approximation formula then becomes:

$$p_1(x) = f(x'_0) + \frac{f(x'_1) - f(x'_0)}{x'_1 - x'_0} (x - x'_0) \qquad (5\text{-}1)$$

Comparing it with the previous formula for Legendre interpolation

$$p_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0)$$

It appears that we have lost several advantages from a hardware implementation standpoint:

$x - x_0$ was directly represented by the $n - n_0$ low order bits of x.

$x_1 - x_0$ had the value 2^{-n_0}

computation of p(x) involved only the multiplication

$$[f(x_1) - f(x_0)] \times R$$

However, equation(5-1)can be further transformed by the following relation:

$$x'_1 - x'_0 = \alpha h$$

$$x - x'_0 = x - (x_0 + \frac{h}{2} - \alpha\frac{h}{2})$$

$$= x - x_0 - \frac{h}{2}(1-\alpha)$$

$$= r - \frac{h}{2}(1-\alpha) = h[R - \frac{1}{2}(1-\alpha)]$$

Thus, given R, one can perform the subtraction [with $(n-n_0)$ bits precision] of the constant $\frac{1}{2}(1-\alpha)$.

Complications arise when the multiplier is not a signed number. It is better to express (5-1) as a function of $r = x - x_0$:

$$p_1(x) = f(x'_0) + \frac{f(x'_1) - f(x'_0)}{x'_1 - x'_0}(x-x_0+x_0-x'_0)$$

$$f_1(x) = f(x'_0) + \frac{f(x'_1) - f(x'_0)}{x'_1 - x'_0}(x_0-x'_0) + \frac{f(x'_1) - f(x'_0)}{x'_1 - x'_0}$$

$$(x-x_0)$$

$$= f(x'_0) + \frac{xf(x'_1) - f(x'_0)}{\alpha h}(\alpha-1)\frac{h}{2} + \frac{f(x'_1) - f(x'_0)}{\alpha h}$$

$$(x-x_0)$$

$$p_1(x) = f(x'_0) + [f(x'_1) - f(x'_0)]\left(\frac{\alpha-1}{\alpha}\right) + \frac{(f(x'_1) - f(x'_0))}{\alpha}R$$

Thus one can store $f(x'_0) + [f(x'_1) - f(x'_0)]\left(\frac{\alpha-1}{2\alpha}\right)$ and $\frac{(f(x'_1) - f(x'_0))}{\alpha}$ in ROMs (or RAMs) and perform the multiplication by R, the $n - n_0$ low order bits of x.

The error for the case $f(x) = \frac{1}{x}$ is $\frac{1}{x} - p_1(x)$ and is tabulated for

$$x_0 = .5 \text{ (first interval)}$$

and $x_0 = 1 - 2^{-11}$ (last interval) in the Chebychev case $(\alpha=\frac{\sqrt{2}}{2})$

and the Legendre case ($\alpha=\frac{\sqrt{3}}{3}$) and for two intermediary cases: $\alpha = .5$ and $\alpha = 1$ in Tables 5.1, 5.2 and 5.3. The $\frac{1}{2}$ case simplifies the hardware design as to the amount of ROM required and the $\alpha = 1$ case corresponds to the Lagrange polynomial for which the formula becomes:

$$p_1(x) = f(x'_0) + [f(x'_1) - f(x'_0)]R.$$

Precision Attainable

$$\frac{1}{x} - p_1(x) = \frac{(x-x'_0)(x-x'_1)}{2!} \times \frac{1}{\xi 3}$$

where ξ takes values between $\xi = x_0$ and $\xi = x_1$.

The polynomial $\pi(x)$ is weighted by a function of x taking values between $\frac{1}{x_0^3}$ and $\frac{1}{x_1^3}$. The error is then bounded by

$$E_1 = \max|\pi(x)| \times \frac{1}{x_0^3} \text{ in the interval } [x_0, x_1].$$

For the Tchebycheff case one finds:

$$E_1 = \frac{(\frac{1}{\sqrt{2}})^2}{4} \frac{1}{x_0^3} k^2.$$

For the first interval: $E_1 = h^2 = 2^{-2n_0}$.

For the last interval: $E_1 = \frac{k^2}{8} = 2^{-2n_0 - 3}$ and thus

x	$1/x - p_i(x)$
0.5000000000000000	0.0000000393633290
0.5000076293945312	0.0000000749337312
0.5000152587890625	0.0000000614354136
0.5000228881835937	0.0000000488683334
0.5000305175781250	0.0000000372324478
0.5000381469726562	0.0000000265277142
0.5000457763671875	0.0000000167540901
0.5000534057617187	0.0000000079115330
0.5000610351562500	0.0
0.5000686645507812	-0.0000000069805515
0.5000762939453125	-0.0000000130301638
0.5000839233398437	-0.0000000181488800
0.5000915527343750	-0.0000000223367422
0.5000991821289062	-0.0000000255937933
0.5001068115234375	-0.0000000279200758
0.5001144409179687	-0.0000000293156326
0.5001220703125000	-0.0000000297805056
0.5001296997070312	-0.0000000293147380
0.5001373291015625	-0.0000000279183723
0.5001449584960937	-0.0000000255914510
0.5001525878906250	-0.0000000223340164
0.5001602172851562	-0.0000000181461115
0.5001678466796875	-0.0000000130277789
0.5001754760742187	-0.0000000069790609
0.5001831054687500	0.0
0.5001907348632812	0.0000000079093607
0.5001983642578125	0.0000000167489793
0.5002059936523437	0.0000000265188127
0.5002136230468750	0.0000000372188187
0.5002212524414062	0.0000000488489544
0.5002288818359375	0.0000000614091775
0.5002365112304687	0.0000000748994453

$\alpha = 1/2$

$h = 2^{-12}$

$h^2 = 0.0000000596046412$

Table 5.1

Error for the First Order Approximation
for $f(x) = 1/x$ and $\alpha = 1/2$

x	$1/x - p_i(x)$
0.5000000000000000	0.0000000595755536
0.5000076293945312	0.0000000451464104
0.5000152587890625	0.0000000316485471
0.5000228881835937	0.0000000190819212
0.5000305175781250	0.0000000074464899
0.5000381469726562	-0.0000000032577894
0.5000457763671875	-0.0000000130309592
0.5000534057617187	-0.0000000218730620
0.5000610351562500	-0.0000000297841407
0.5000686645507812	-0.0000000367642379
0.5000762939453125	-0.0000000428133959
0.5000839233398437	-0.0000000479316578
0.5000915527343750	-0.0000000521190657
0.5000991821289062	-0.0000000553756625
0.5001068115234375	-0.0000000577014907
0.5001144409179687	-0.0000000590965932
0.5001220703125000	-0.0000000595610119
0.5001296997070312	-0.0000000590947900
0.5001373291015625	-0.0000000576979700
0.5001449584960937	-0.0000000553705943
0.5001525878906250	-0.0000000521127055
0.5001602172851562	-0.0000000479243463
0.5001678466796875	-0.0000000428055593
0.5001754760742187	-0.0000000367563870
0.5001831054687500	-0.0000000297768719
0.5001907348632812	-0.0000000218670568
0.5001983642578125	-0.0000000130269839
0.5002059936523437	-0.0000000032566962
0.5002136230468750	0.0000000074437640
0.5002212524414062	0.0000000190743541
0.5002288818359375	0.0000000316350315
0.5002365112304687	0.0000000451257536

$\alpha = \sqrt{2}/2$

$h = 2^{-12}$

Table 5.2

Error for the First Order Approximation for $f(x) = 1/x$ and $\alpha = \sqrt{2}/2$ (Tchebycheff approximation)

x	$1/x-p_1(x)$
0.5000000000000000	0.0000000794340707
0.5000076293945312	0.0000000650046246
0.5000152587890625	0.0000000515064584
0.5000228881835937	0.0000000389395296
0.5000305175781250	0.0000000273037954
0.5000381469726562	0.0000000165992133
0.5000457763671875	0.0000000068257406
0.5000534057617187	-0.0000000020166651
0.5000610351562500	-0.0000000099280466
0.5000686645507812	-0.0000000169084466
0.5000762939453125	-0.0000000229579076
0.5000839233398437	-0.0000000280764723
0.5000915527343750	-0.0000000322641831
0.5000991821289062	-0.0000000355210827
0.5001068115234375	-0.0000000378472138
0.5001144409179687	-0.0000000392426192
0.5001220703125000	-0.0000000397073407
0.5001296997070312	-0.0000000392414217
0.5001373291015625	-0.0000000378449045
0.5001449584960937	-0.0000000355178318
0.5001525878906250	-0.0000000322602458
0.5001602172851562	-0.0000000280721895
0.5001678466796875	-0.0000000229537054
0.5001754760742187	-0.0000000169048360
0.5001831054687500	-0.0000000099256237
0.5001907348632812	-0.0000000020161115
0.5001983642578125	0.0000000068236585
0.5002059936523437	0.0000000165936433
0.5002136230468750	0.0000000272938008
0.5002212524414062	0.0000000389240880
0.5002288818359375	0.0000000514844625
0.5002365112304687	0.0000000649748817

$\alpha = \sqrt{3}/3$

$h = 2^{-12}$

Table 5.3

Error for the First Order Approximation for $f(x) = 1/x$ and $\alpha = \sqrt{3}/3$ (Legendre approximation)

$$2^{-2n_0-3} \leq \text{Error} \leq 2^{-2n_0}$$

The use of the zeros of the Tchebycheff polynomials leads to the minimum value of $|\pi(x)|max$ and yields a value smaller than in the Legendre case. If it is desirable to control the maximum error, the Tchebycheff polynomials are preferable.

The zeros of the Legendre polynomial will minimize the RMS value of $\pi(x)$ over the interval.

5.3 Second Order Interpolation

The process is the same as for linear interpolation except that $\frac{1}{x}$ will be approximated by a second degree polynomial in each interval.

The collocation points are found to be, as before, the roots of a Tchebycheff or Legendre polynomial:

$$\underline{\text{Legendre}}: \quad P_3(x) = \frac{1}{2}x\,(5x^2-3) \text{ with roots: } -\sqrt{\frac{3}{5}},\ 0,\ \sqrt{\frac{3}{5}}$$

$$\underline{\text{Tchebychev}}: \quad T_3(x) = x(4x^2-3) \text{ with roots: } -\sqrt{\frac{3}{4}},\ 0,\ \sqrt{\frac{3}{4}}$$

The roots are of the form: $-\alpha,\ 0,\ +\alpha$.

The change of variable to map $[-1, +1]$ into x_0, x_1 leads to the following collocation points:

$$x'_1 = -\alpha\,\frac{h}{2} + \frac{x_1 + x_0}{2}$$

$$x'_2 = \alpha\,\frac{h}{2} + \frac{x_1 + x_0}{2}$$

$$x'_3 = \frac{x_1 + x_0}{2}.$$

Let us now derive the equation in a form suitable for hardware implementation. One would like a formula of the type:

$p_2(x) = a_0 + a_1R_1 + a_2R^2$ where R is the number defined previously. We have

$$p_2(x) = \frac{(x-x'_2)(x-x'_3)}{(x_1-x'_2)(x_1-x'_3)}\, y_1 + \frac{(x-x'_1)(x-x'_3)}{(x'_2-x'_1)(x'_2-x'_3)}\, y_2 + \frac{(x-x'_1)(x-x'_2)}{(x'_3-x'_1)(x'_3-x'_2)}\, y_3.$$

After some elementary transformations, one obtains the polynomial:

$$p_2(x) = \left(y_1\frac{(1+\alpha)}{2\alpha^2} - \frac{(1-\alpha^2)}{\alpha^2}\, y_2 + \frac{(1-\alpha)}{2\alpha^2}\, y_3\right) + \left(\frac{-(2+\alpha)\, y_1 + 4y_2 - (2-\alpha)\, y_3}{\alpha^2}\right) R + \left(\frac{2y_1 - 4y_2 + 2y_3}{\alpha^2}\right) R^2$$

which can be evaluated in several different ways (e.g., parallel multiplication or Horner's rule).

The precision can be calculated as follows: We have

$$f(x) - p(x) = \frac{(x-x'_1)(x-x'_2)(x-x'_3)}{3!}\, f'''(\xi).$$

For the division case,

$$\frac{1}{x} - p(x) = (x-x'_1)(x-x'_2)(x-x'_3) \times \frac{1}{\xi 4}.$$

In the Tchebychev case $\pi(x)$ is bounded by:

$$E = \frac{a^3h^3}{12.\sqrt{3}} \text{ with } \alpha = \frac{3}{4} \text{ or}$$

$$E = \frac{3\sqrt{3}h^3}{8x12x\sqrt{3}} = \frac{h^3}{2^5}$$

The upper bound on the error is,for the first interval $[.5, .5 + 2^{-n_0}]$

$$E = \frac{h^3}{2^5} \cdot 2^4 = \frac{h^3}{2^5}$$

and for the last interval $E = \frac{h^3}{2^5}$ and thus

$$2^{-3n_0-5} \leq \text{Error} \leq 2^{-3n_0-1} \qquad n_0 = \text{number of bits looked up.}$$

5.4 Higher Order Interpolation and Precision

For interpolations of order higher than 3 it is possible to obtain speed improvement by using any of the well-known "economical polynomial evaluations". There are two sources of error when a polynomial is evaluated. For example in evaluating:

$$p_1(x) = a_0 + a_1R,$$

the two errors are:

1) $f(x) - p(x) = \frac{(x-x'_0)(x-x'_1)}{2} f''(\xi) = E_1$ and

2) the representation of numbers using finite length registers and multipliers.

In fact one performs

$$p_1(x) = (a_0 \pm E_0) + (a_1 \pm E_1)(R \pm E_R)$$

and the maximum error due to the adder is E_0.

The maximum error due to the finite length inputs to the multiplier is:

$$E_1R + E_Ra_1 + E_1E_R \leq E_1 + E_R + E_1E_R$$

One can reduce E_0 relative to $(2E_1+E_1^2)$ by using more accuracy for the first constant a_0.

The other source of error is then due to the multiplier. The task of the designer is to optimize the trade-off between time and precision. Totally parallel evaluation is expensive and requires a large multiplier whereas semi-parallel multiplication can make use of a smaller multiplier but is more time consuming.

For example using a 24 x 24 bit multiplier, is it possible to have a division routine requiring one multiplication cycle time for a precision of 2^{-24} or less?

By using a first order optimal interpolation and looking up 11 bits of the number x, the upper bound of the division routine error will be:

$$2^{-2n_0} = 2^{-24}.$$

The formula implemented is

$p_1(x) = a_0 + a_1R$. The error on a_1 and R is called E_1. If $E_1 = 2^{-25}$ the errors can add giving a total error of 2^{-23}. The only solution is to have

a) a bigger multiplier, making the error due to the series truncation predominate or

b) for the same multiplier, use smaller intervals and thus make the error due to the multiplier the predominate.

In the example, if 12 bits of the initial number are looked-up, the error due to truncation of the series is bounded by

$$2^{-2n_0} = 2^{-26}.$$

The predominant error is due to the multiplication,

$$a_1 x R,$$

and is 2^{-24} because of the 24 x 24 bit multiplier.

5.5 Application to Arbitrary Function Generation in Aviation

Aircraft and engine performance data is received in the form of plots of performance variables as functions of other system variables. The functional relationships generally are multivariable and nonlinear, and can seldom be approximated by analytical equations without introducing inaccuracies beyond the specification tolerances. Furthermore, the data is subject to change as experience is gained with the aircraft--generally in the direction of increasing complexity.

At present all function generation problems are solved by software and, as a result, the analytical approximations such as polynomial expansions must also increase in complexity, resulting in a corresponding increase in simulation program execution time.

Function generation by linear interpolation of stored data points is generally superior in aviation to analytical approximations for two reasons:

1. Data can be readily programmed directly from plots or tables without extra design time to develop an approximation;
2. Increased data complexity can be met with no increase in execution time, since the interpolation formula does not change.

Notations:

$f(x_1)$ and $f(x_2)$ are adjacent function values

x_1 and x_2 are the corresponding argument values

x is the current argument value

h the difference $x_2 - x_1$

Since many functions can use the same breakpoints or argument values a saving in time and memory space may be realized if the "interpolant"

$$\frac{x - x_i}{x_{i+1} - x_i}$$

is shared by the functions. (Typically the interpolant may be shared by three or more functions in practical cases.)

If fast multiplication and division hardware routines are available, the majority of the time consumed in the interpolation process will be due to the "argument search" routine (which can be implemented in hardware too).

The linear interpolation for one variable,

$$f(x) = [f(x_{i+1}) - f(x_i)] \frac{x - x_i}{x_{i+1} - x_i} + f(x_i),$$

requires

1. Argument search: given x, where is x_i?
2. One division.
2. One multiplication.

A schematic of the organization of the AU is given next page.

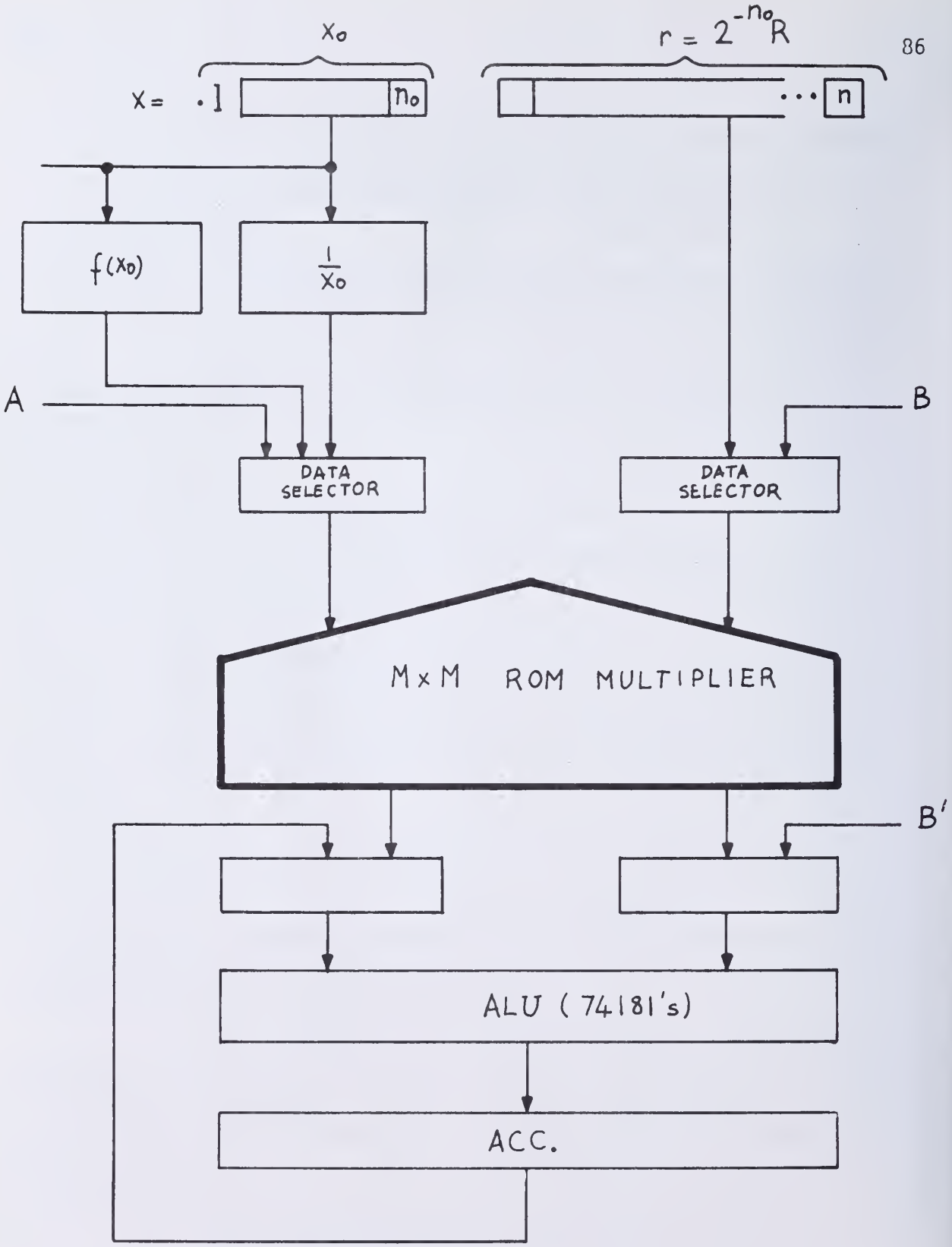


Figure 5.1 Organization of an ALU for function generation

6. CONCLUSION

The purpose of this investigation was to explore various methods for generating functions for processors used in aircraft simulators. A small 8-bit CORDIC prototype with microprogrammed control was constructed implementing the trigonometric mode. Other methods based on polynomial approximations have also been reviewed. The CORDIC method has the advantage of simplicity in implementation and, being a bit by bit method using only addition and shifting as the basic operations, it can be cheaper to implement than a parallel method. For aviation simulation,it is a matter of choice between the following alternatives:

a) A central arithmetic unit performing all the computations and being time-shared between all the flight instruments. In this case, obtaining a frequent updating of the instruments requires a fast arithmetic unit.

b) Several arithmetic units doing specialized tasks (altitude computation, speed, fuel, ... Each AU could be a relatively slow and inexpensive bit-by-bit serial AU.

The design of an aircraft simulation system would use a top-down development process. Since the simulator in the decentralized architecture would contain more than one arithmetic unit, the development would proceed such that modules could be built and tested as soon as they are developed.

Operations requiring simple and infrequent computations would be implemented by CORDIC bit serial algorithms.

Operations requiring frequent and complex computations (like function generation in an aircraft) would use polynomial approximation algorithms for faster execution.

Polynomial approximation methods have been studied which make use of polynomials other than the Taylor-MacLaurin series currently used in hardware implementations for division. It has been shown that better error control is achieved by using Tchebychef polynomials. The advantage of the parallel algorithms is that they employ multiplicative operations which can share a single n x n multiplier.

The potential parallelism of these algorithms should be further investigated. For example, it is known that a polynomial of degree n can be evaluated in O(logn) operations. Is it possible to design (in an array expandable form) an n x n x n multiplier or more generally an n^k multiplier? By implementing parallelism at the gate level (which could be called microparallelism), what improvements can be achieved for special purpose highly parallel and high speed arithmetic processors?

Since polynomial evaluation lends itself to parallel processing, what would the time bound on the arithmetic computation of functions become? What improvements would be achieved by replacing the processors performing binary operations by a 3- or k- processor performing k-nary multiplications and additions?

Function evaluation by hardware means is currently being further investigated as well as some possible designs for an n^3 multiplier.

7. Appendix

```
NUMBER ØF BITS =?20

M = ?1

TEST SIGN =?Y

X--> K1*SQRT(X0**2+Y0**2)
Y-->0
Z-->Z0 + ARCTAN(Y0/X0)

K1=1.646760258121Ø6
1/K1=0.6Ø725293500888

X0 = ?.6072529

Y0 = ?-.6072529

-180 DEG.<= Z0 <= +180 DEG

Z0 = ?0

INIT. ØK  ?Y

DECIMAL ØNLY ?N

---

 0.6072502136   0 0.1001101101110011        SUB
                0 0.0000000000000000

                0 0.0000000000000000
-0.6072502136   1 1.0110010010001101        ADD

 0.0            0 0.0000000000000000        SUB
90.0000000000   0 1.0000000000000000
 INITIALIZATIØN

---
```

A.1 Output for the Trigonometric Mode

```
     0.6072502136   0 0.1001101101110100 11        ADD
                    0 0.1001101101110100 11

                    0 0.1001101101110100 11
     0.6072502136   0 0.1001101101110100 11        SUB

   -90.0000000000   1 1.0000000000000000 00        ADD
    45.0000000000   0 0.1000000000000000 00
     FIRST 90 DEG. RØTATIØN

---

     1.2145004272   0 1.0011011011101001 10        ADD
                    0 1.0011011011101001 10

                    0 0.0000000000000000 00
     0.0            0 0.0000000000000000 00        SUB

   -45.0000000000   1 1.1000000000000000 00        ADD
    26.5649414063   0 0.0100101110010000 00
    # SHIFTS =  0

---

     1.2145004272   0 1.0011011011101001 10        SUB
                    0 0.1001101101110100 11

                    1 1.1011001001000101 10
    -0.6072502136   1 1.0110010010001011 01        ADD

   -18.4350585938   1 1.1100101110010000 00        SUB
    14.0360641479   0 0.0010011111101100 11
    # SHIFTS =  1
```

A.1 (cont.)

```
     1.4142265320   0 1.011010100000101011        SUB
                    0 0.000000000000000010

                    1 1.111111111111111111

    -0.0000038147   1 1.111111111111111111        ADD

   -45.0006866455   1 1.011111111111111110        SUB
     0.0            0 0.000000000000000000
    # SHIFTS = 17

---

     1.4142303467   0 1.011010100000101100        ADD
                    0 0.000000000000000001

                    0 0.000000000000000000
     0.0            0 0.000000000000000000        SUB

   -45.0006866455   1 1.011111111111111110        ADD
     0.0            0 0.000000000000000000
    # SHIFTS = 18
```

CØRDIC VALUES	TRUE VALUES	ERRØR IN %
1.414230346680	1.414163948760	0.00470
0.0	0.0	**********
-45.000686645508	-45.000000000000	0.00153

A.1 (cont.)

```
NUMBER ØF BITS =?20

M = ?0

TEST SIGN =?Y

X-->X0
Y-->0
Z-->Z0 + (Y0/X0)

X0 = ?.6

Y0 = ?-.3

Z0 = ?0

INIT. ØK  ?Y

DECIMAL ØNLY ?N

---

 0.5999984741   0 0.10011001100110011010        SUB
                0 0.00000000000000000000

                0 0.00000000000000000000
-0.2999992371   1 1.10110011001100110101        ADD

 0.0            0 0.00000000000000000000        SUB
 0.5000000000   0 0.10000000000000000000
 INITIALIZATIØN
```

A.2 Output for the Linear Mode

```
---

     0.5999984741   0 0.10011001100110011        SUB
                    0 0.01001100110011001

                    0 0.00000000000000000
     0.0            0 0.00000000000000000        SUB

    -0.5000000000   1 1.10000000000000000        ADD
     0.2500000000   0 0.01000000000000000
    # SHIFTS =  1

---

     0.5999984741   0 0.10011001100110011        SUB
                    0 0.00100110011001100

                    1 1.11110110011001100
    -0.1499977112   1 1.11011001100110011        ADD

    -0.2500000000   1 1.11000000000000000        SUB
     0.1250000000   0 0.00100000000000000
    # SHIFTS =  2
```

A.2 (cont.)

```
 0.5999984741   0 0.1001100110011001l0        SUB
                0 0.0000000000000l0011

                1 1.1111111111111111

-0.0000915527   1 1.1111111111101000         ADD

-0.4998779297   1 1.1000000000100000         SUB
 0.0000610352   0 0.0000000000010000
# SHIFTS = 13

 0.5999984741   0 0.1001100110011001l0        SUB
                0 0.0000000000001001

                1 1.1111111111111111

-0.0000572205   1 1.1111111111110001         ADD

-0.4999389648   1 1.1000000000010000         SUB
 0.0000305176   0 0.0000000000001000
# SHIFTS = 14

 0.5999984741   0 0.1001100110011001l0        SUB
                0 0.0000000000000000

                1 1.1111111111111111

-0.0000305176   1 1.1111111111111000         ADD

-0.4999961853   1 1.1000000000000001         SUB
 0.0            0 0.0000000000000000
# SHIFTS = 19
```

CØRDIC VALUES	TRUE VALUES	ERRØR IN %
0.599998474121	0.600000000000	0.00025
-0.000030517578	0.0	**********
-0.499996185303	-0.500000000000	0.00076

A.2 (cont.)

```
NUMBER ØF BITS =?20

M = ?-1

TEST SIGN =?Z

X-->K2*(XØ*CØSH ZØ + YØ*SINH ZØ)
Y-->K2*(YØ*CØSH ZØ + XØ*SINH ZØ)
Z-->0

K2 = 0.82978162013890
1/K2 = 1.20513635844646

XØ = ?1.205136

YØ = ?0

-180 DEG.<= ZØ <= +180 DEG

ZØ = ?-45

INIT. ØK  ?Y

DECIMAL ØNLY ?N

---

     1.2051353455   0 1.0011010010000011111        SUB
                    0 0.0000000000000000000
                    0 0.0000000000000000000
     0.0            0 0.0000000000000000000        SUB
   -45.0000000000   1 1.1000000000000000000        ADD
    49.4374389648   0 0.1000110010011111101
     INITIALIZATIØN
```

A.3 Output for the Hyperbolic Mode

```
---

     1.2051353455   0 1.00110100100000111 1        ADD
                    0 0.10011010010000011 1

                    1 1.10110010110111110 0
    -0.6025657654   1 1.01100101101111100 1        ADD

     4.4374465942   0 0.00001100100111110 1        SUB
    22.9868316650   0 0.01000001011000101 0
    # SHIFTS =  1

---

     1.0544929504   0 1.00001101111100110 1        SUB
                    0 0.01000011011111001 1

                    1 1.11101100101101111 1
    -0.3012847900   1 1.10110010110111110 0        SUB

   -18.5493850708   1 1.11001011001111001 1        ADD
    11.3090515137   0 0.00100000001010110 0
    # SHIFTS =  2
```

A.3 (Cont.)

```
 1.1254158020   0 1.00100000000101101        ADD
                0 0.000000000000000001

                1 1.111111111111111111
-0.5200729370   1 1.011110101101110010        ADD

 0.0003433228   0 0.000000000000000001        SUB
 0.0            0 0.000000000000000000
# SHIFTS = 18

---

 1.1254119873   0 1.00100000000101100         ADD
                0 0.000000000000000000

                1 1.111111111111111111
-0.5200729370   1 1.011110101101110010        ADD

 0.0003433228   0 0.000000000000000001        SUB
 0.0            0 0.000000000000000000
# SHIFTS = 19
```

CØRDIC VALUES	TRUE VALUES	ERRØR IN %
1.125411987305	1.127625629814	0.19631
-0.520072937012	-0.521095150503	0.19617
0.000343322754	0.0	**********

A.3 (Cont.)

Appendix A.4 Overview of the Algorithms Proposed By Baker, Ferrari and Ling

P.W. Baker [BAK75] presents generalized higher radix algorithms for some elementary functions. Parallelism is achieved by using fast parallel m-bit multipliers where radix = 2^m. They are a generalization of the CORDIC and deLugish algorithms based on multiplications by prestored constants of the form (1 ± 2^{-i}) or $\ln(1 \pm 2^{-i})$.

Baker's algorithm for Y/X uses

$$Q = \frac{Y}{X} = \frac{Y \prod f_i}{X \prod f_i}$$

where the f_i are selected to be of the form

$$(1+d_i\, r^{-k}) \qquad d_i \varepsilon \{0,1,2\ldots,r-1\}$$

the algorithm is

$$Y_{i+1} = X_i(1+d_i r^{-k})$$

$$Y_{i+1} = Y_i(1+d_i r^{-k})$$

The first multiplier d_1 is chosen by table look-up. Then at each step d_i is chosen as the one's complement of a_k, a_k being the k^{th} 2^m-ary coefficient of the number x_i written base 2^m. The hardware requires variable shift networks as in the CORDIC algorithms and, in addition, two m-bits multipliers.

Ferrari uses a Lagrange interpolation method of the function $\frac{1}{x}$ between x = .5 and x = 1. He uses several colloquation points and looks up the coefficients of the first order interpolating polynomial.

Ling proposes a high-speed division algorithm based, like Steffanelli's on the Taylor expansion of a fraction of the form $\frac{1}{1+x}$. The eight leading digits of the denominator are looked up and two multipliers working in parallel are needed to evaluate the approximating polynomial. This algorithm is designed for 32 bit precision and the same criticisms hold for both Stefanelli's and Ling's algorithms.

A.5 ROM Usage As Proposed By Stefanelli

Stefanelli proposes a "binary read only memory divider". He shows the realisation of three inverting circuits computing the reciprocal $Q = \frac{1}{B}$ of a binary number B. Their main feature is the use of read-only memories whose function are to provide look-up of prestored polynomial coefficients, and polynomial values. The three circuits differ only in the number of bits of the binary number B.

The number B is normalized

$$B = 1.\, b_1 b_2 \ldots b_r\, b_{r+1} \ldots b_m$$

and the reciprocal is

$$Q = 0.1,\, q_1 q_2 \ldots q_m.$$

The idea is to split B into two parts B_1 and B_2:

$$B = B_1 + 2^{-r} B_2$$

and look up the inverse Q_1 of B_1:

$$\frac{1}{B_1} = Q_1 = 0.1\, q'_1\, q'_2 \ldots q'_m$$

Then

$$Q = \frac{1}{B} = \frac{1}{B_1 + 2^{-r}B_2} = \frac{1}{B_1} \frac{1}{1 + 2^{-r} \frac{B_2}{B_1}}$$

$$= Q_1 \frac{1}{1 + 2^{-r}(B_2 Q_1)}.$$

The fraction is expandable in a Taylor series

$$\frac{1}{1+x} = 1 - x + x^2 - x^3 \ldots$$

$$Q = Q_1(1-2^{-r}Q_1B_2+2^{-2r}(Q_1B_2)^2 - 2^{-3r}(Q_1B_2)^3 + \ldots)$$

Now Q is to be evaluated using a three-stage procedure:

1 - Look Up

2 - Multiplication

3 - Addition/Subtraction.

With current technology r can be chosen to be up to 14 (16K of memory addresses). If, for example, the first term of the series is considered, i.e.

$$Q = Q_1 - 2^{-r} Q_1^2 B_2$$

The error arises primarily

1) From the truncation error of the series, whose absolute value is known to be less than the maximum value of the first term neglected. Here:

$$\delta_1 \leq 2^{-2r}$$

2) From the computation of $2^{-r}Q_1^2B_2$

if B_2 is supposed to be known with any desirable precision, the error remaining is due to Q_1^2. The number of bits of Q_1^2 is chosen such that this error is of the same order as the truncation error.

When a higher number of bits are required, the second term of the series is considered:

$$Q = Q_1 - 2^{-r} Q_1^2 B_2 + 2^{-2r} Q_1^3 B_2^2.$$

However, B_2 can be further split into two parts ,

$$B_2 = B_3 + 2^{-S} B_4 \quad ,$$

in order to avoid too many bits of address for the computation (or look-up of B_2^2). The term neglected are $Q_1^3 B_4^2$ and $Q_1^3 B_3 B_4$.

The first and second schemes are shown in Figures A.5.1 and A.5.2.

One can criticize the method in that it makes use of the Taylor polynomial for approximation. Other polynomials exist having better error behavior for the same cost in terms of ROM requirements and same speed. Also, a set of small dedicated multipliers is not very efficient because they cannot be used for any other functions or even for multiplication of two full-size numbers. It is also impossible to use the same topology for sequential evaluation of higher precision division (more than 32 bits).

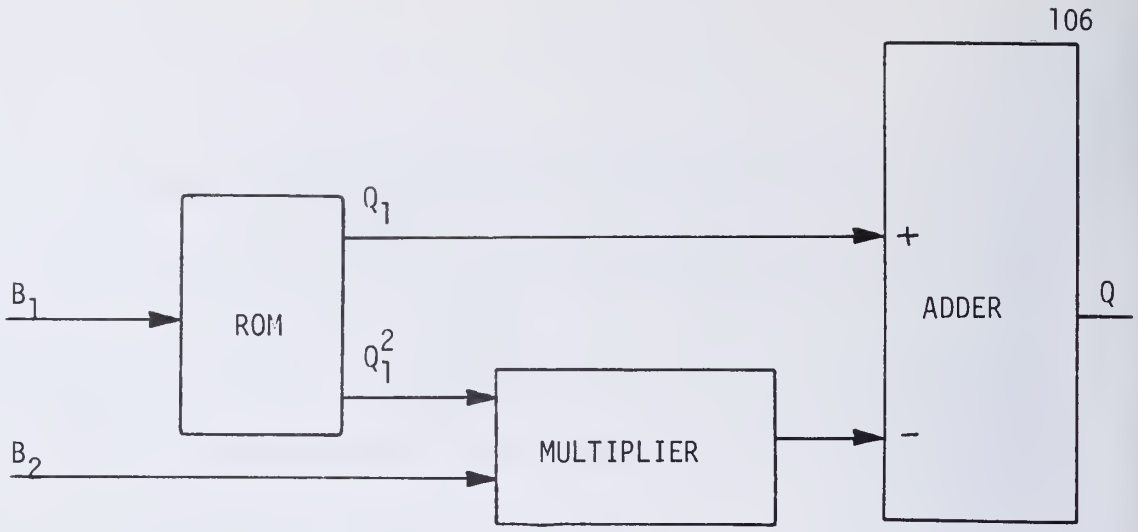


Figure A.5.1

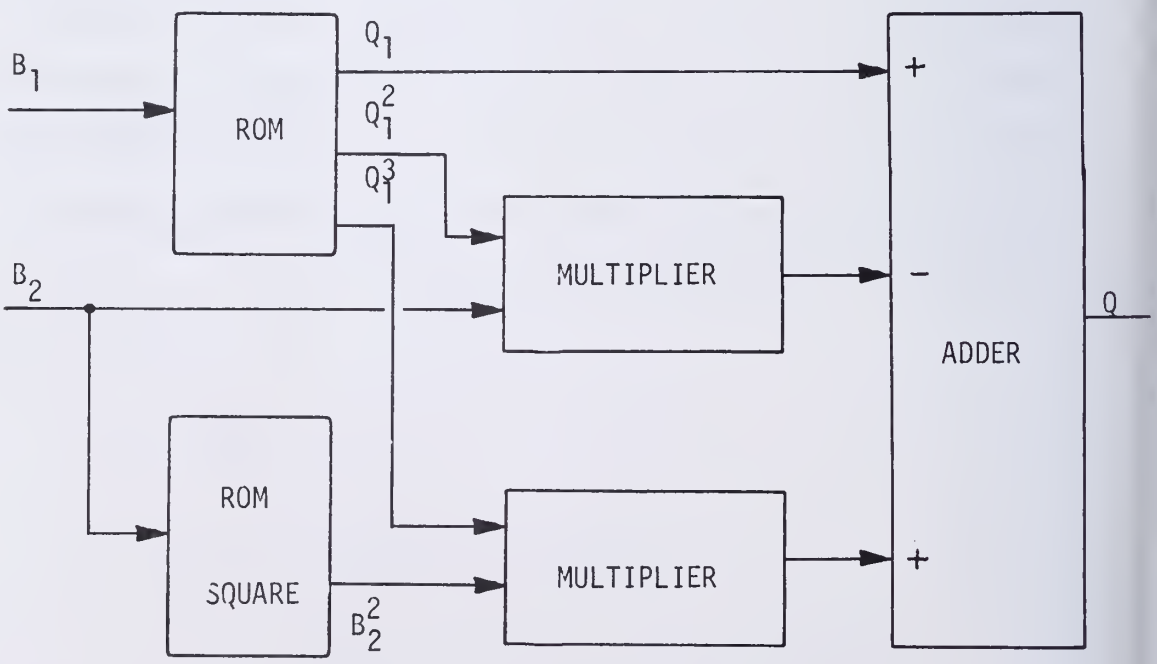


Figure A.5.2 ROM Usage proposed by Steffanelli

8. REFERENCES

[BAK75] Baker, P. W. "Parallel Multiplicative Algorithms for Some Elementary Functions". IEEE Transactions on Computers, March 1975, pp. 3.

[CAN70] Cantor, D., Estrin, G. E. and Turn, R."Logarithmic and exponential function generation in a variable structure digital computer, IRE Trans. on Electronic Digital Computer, Vol EC-11, Apr 1962.

[CHE72] Chen, T. "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots". IBM J. Res. Development, Vol. 16, July 1972, pp. 380-8."

[FER71] Ferrari, D. "A Division Method using a Parallel Multiplier" IEEE Transactions on Electronic Computers, April 1967, pp. 224-228.

[LIN71] Ling, H. "High Speed Division for Binary Computers". Proc. 1971 Spring Joint Computer Conf., Vol. 38, AFIPS Press, Montvale, N. J., 1971, pp. 373-8.

[LUG70] DeLugish, B. G. "A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer", Ph. D Dissertation. Department of Computer Science, University of Illinois, Urbana, Ill., June 1970.

[STE74] Stefanelli, R. "A High Speed Binary Divider". Unpublished paper submitted to the IEEE repository system.

[WAL64] Wallace, C. S. "A Suggestion for a Fast Multiplier". IEEE Transactions on Electronic Computers, Vol. EC 13, February 1964, pp. 14-17.

[WAL71] Walter, T. S. "A Unified Algorithm for Elementary Functions," AFIPS Conference Proceedings, Spring Joint Computer Conference, pp. 379 to 385, 1971.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-796	2.	3. Recipient's Accession No.

4. Title and Subtitle
SPECIAL PRUPOSE PROCESSORS FOR RADIO AIDS AND FUNCTION GENERATION IN AIRCRAFT SIMULATORS

5. Report Date: February 1976

6.

7. Author(s): GILLES H. GARCIA

8. Performing Organization Rept. No. UIUCDCS-R-76-796

9. Performing Organization Name and Address
University of Illinois at Urbana-Champaign
Department of Computer Science and
Institute of Aviation
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address
Institute of Aviation
University of Illinois
Urbana, Illinois 61801

13. Type of Report & Period Covered: Master's Thesis

14.

15. Supplementary Notes

16. Abstracts

The purpose of this work was to investigate the feasibility of a special purpose hardware processor for flight simulation. The study is centered around the arithmetic part of the processor for use in a real time environment where high accuracy and resolution are required for values of the elementary transcendental functions.

Two main approaches have been considered:

-Coordinate rotation methods (Volder, De Lugish and Walther)

-Methods based on function approximation employing techniques common to Numerical Analysis and relying on algorithms using multiplication as the basic operator.

An 8-bit Arithmetic Cordic Unit has been built and the trigonometric mode microprogrammed. In addition, the general n-bit Arithmetic unit has been simulated for base 2. Errors bounds for some numerical algorithms have been derived.

17. Key Words and Document Analysis. 17a. Descriptors

Function Generation, Arithmetic Unit, Cordic Algorithms, Coordinate Rotation technique, De Lugish Algorithms, Microprogramming.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	117
	20. Security Class (This Page) UNCLASSIFIED	22. Price --